

Report Builder for Advanced Beginners

- From my childhood: “Advanced Beginners” are those kids who
 - have passed *Swimming for Beginners*
 - can swim, a little
 - have trouble going forward.
- Here, I’ll assume you have:
 - opened the *Report Builder* box
 - have trouble actually creating a Report.

We will use Report Builder to:

- Create correct SQL
- Modify that SQL
- Create a complete report
- Modify that report

■ We will craft a Report to:

- count how many Generators, Transporters, TSD, etc., are in each State in a Region
- let the user choose the counting methodology

The SELECT command

- SELECT *column1 , column2...*
- FROM *table1 , table2...*
- WHERE *relationshipExpression*
 - WHERE table1.column2 = table3.column5
 - WHERE column6 = '04'
 - WHERE column3 IN (SELECT column...)
- GROUP BY *column4...*

COUNT() and SUM()

- COUNT(*table3.column8*)
 - *as in:* SELECT column2, COUNT(column4)...
- SUM(*table2.column8*)
 - *as in:* SELECT column5, SUM(column1)...
- Both of these Functions often require that the SELECT command also specify a GROUP BY column.

DECODE()

- The DECODE() Function examines one particular column, and for each row of the output table, it compares that column's value to one or more programmed values.
- If it finds a match, it replaces that row's column-value with another specified value.
- If there is no match found, it still replaces that column-value with a 'default' value.

DECODE() Syntax

- DECODE(column,
match1, replaceValue1,
<optional additional match/replaces,>
defaultReplacementValue)
- For example:
 - DECODE(state, 'GA' , 'HOME' , 'AWAY')
 - DECODE(name, 'BOB', 10, 'DAVE', 20, 0)
- The column's *type* is changed to match the *type of match1*.

Oracle Report Builder

- Report Builder helps us:
 - get data out of Oracle-brand databases
 - lay that data out in a nice format
- When we create our report's layout, we will be arranging (and rearranging):
 - *Labels* (boilerplate text that does not change)
 - *Fields* (text from our SQL output table)
 - *Frames* (invisible-ish boxes that group other items together and control formatting issues.

Two Kinds of Frames: Repeaters and Monads

■ Repeater Frames:

- are connected to a SQL group-by column
- print this way:
 - print everything inside themselves;
 - check to see if there is another unprinted row in their grouping-chunk of the output table;
 - repeat this process if there is.
- Repeaters are auto-named *R_G_groupColumn*
- Builder will create the SQL group for you

Two Kinds of Frames: Repeaters and Monads

■ Monad Frames:

- Print only once, and that's it
- Named for Leibnitz's philosophical *Monad*; they are complete in themselves and "external cause can have no influence upon their inner being." `M_G_groupColumn`
- Of course, if they are inside a Repeater frame, they will be printed as often as the Repeater prints itself.

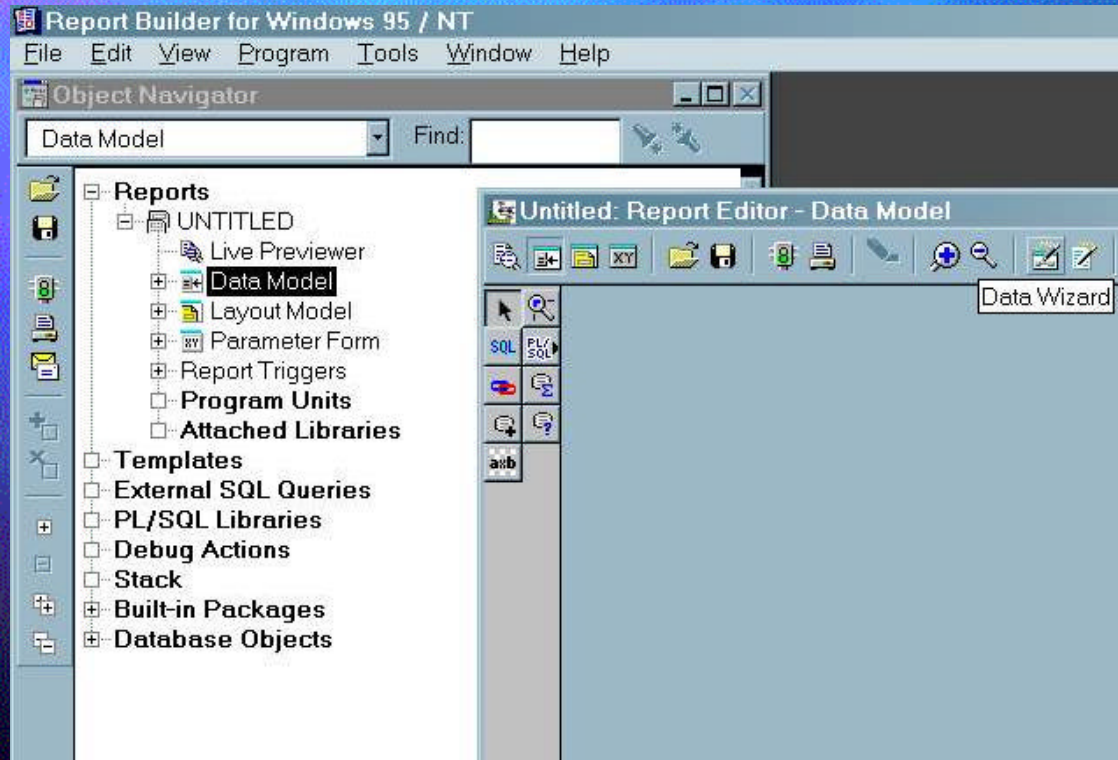
Let's Get Down to Business

- We want to create a report that:
 - asks the user “What Region?”
 - counts up all generators, transporters and others in that Region
 - presents the results by summarizing each kind of count for each State in that Region
- Question: where is this data kept?

Where is the Data?

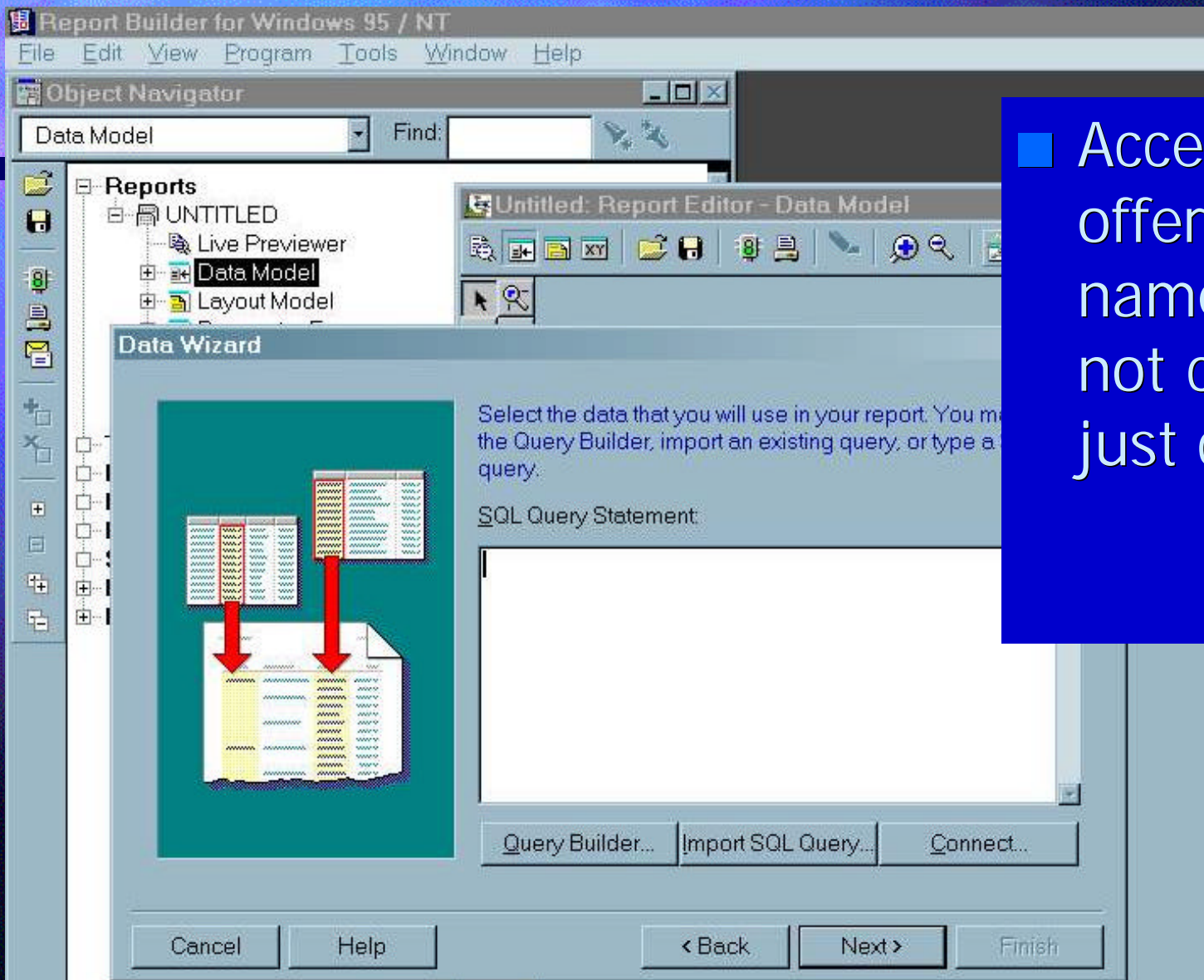
- In HBASIC, we can find Region, State, and Handler IDs
- In HUNIVERSE, we can find FK_LU_UNIVERSEUNIVERSE_TYPE
 - FK... = 'LQG' if that ID is an LQG
 - it also = 'SQG', 'CEG', 'TRANSPORT', 'SUBJINSP', 'SUBJCA' when applicable
 - One ID can have multiple UniverseTypes

So Open Report Builder



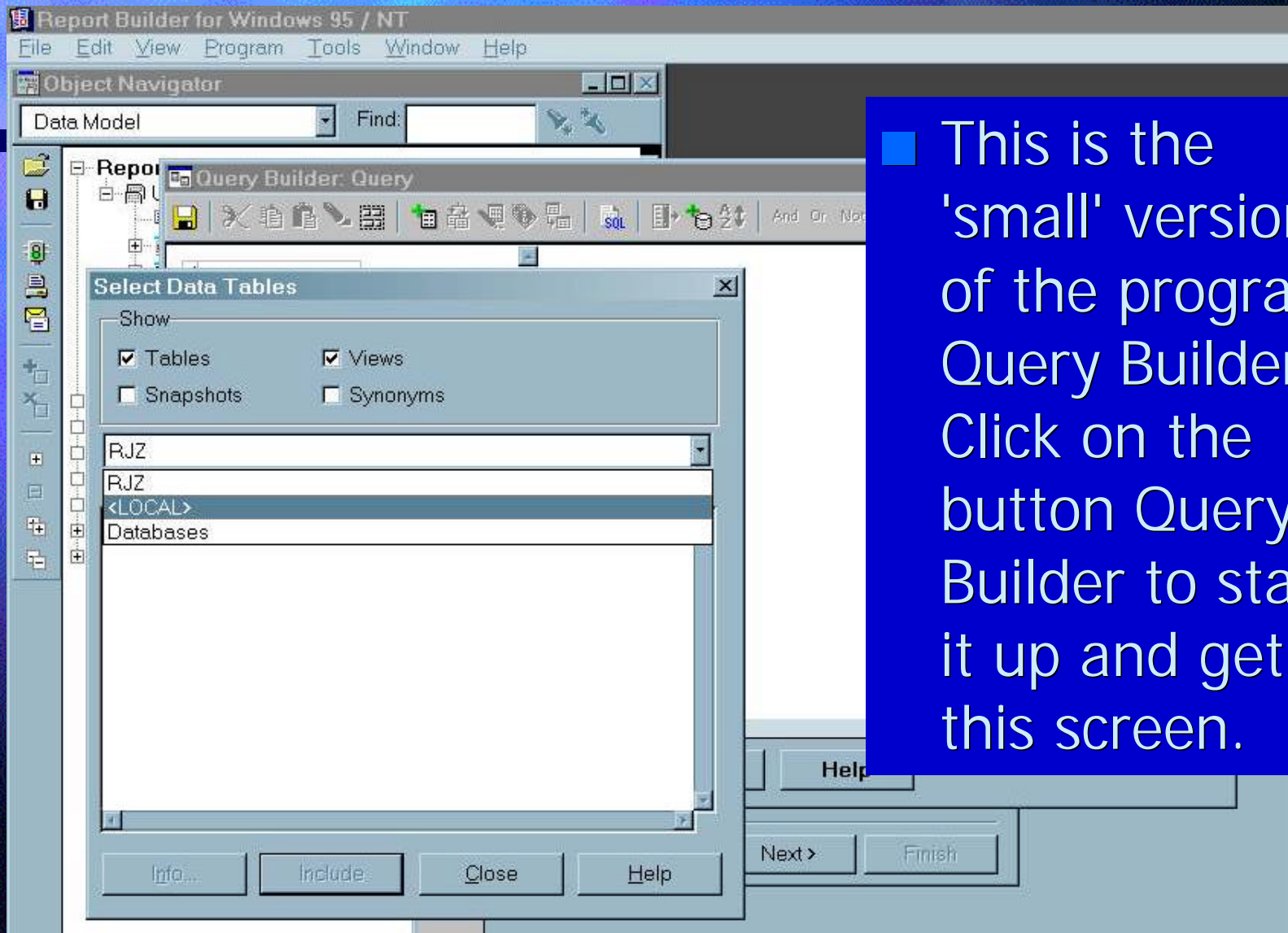
- (decline any offers of Wizard help)
- Double-click on the small icon next to the words Data Model
- Click on the Data Wizard icon (let your mouse linger over the possibilities to be sure which one you want)

Start the Data Wizard



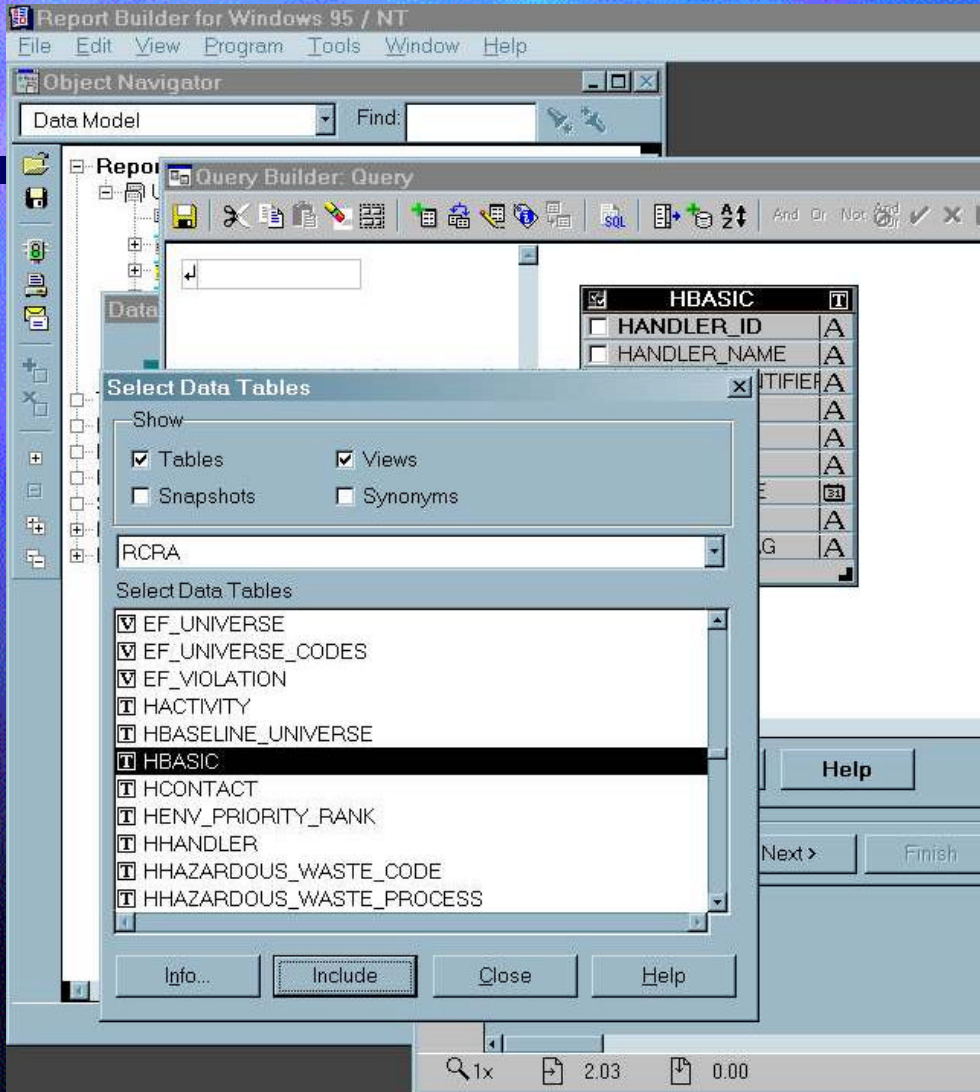
- Accept the offered query name, and do not click Matrix; just click Next.

Start Query Builder



- This is the 'small' version of the program Query Builder. Click on the button Query Builder to start it up and get this screen.

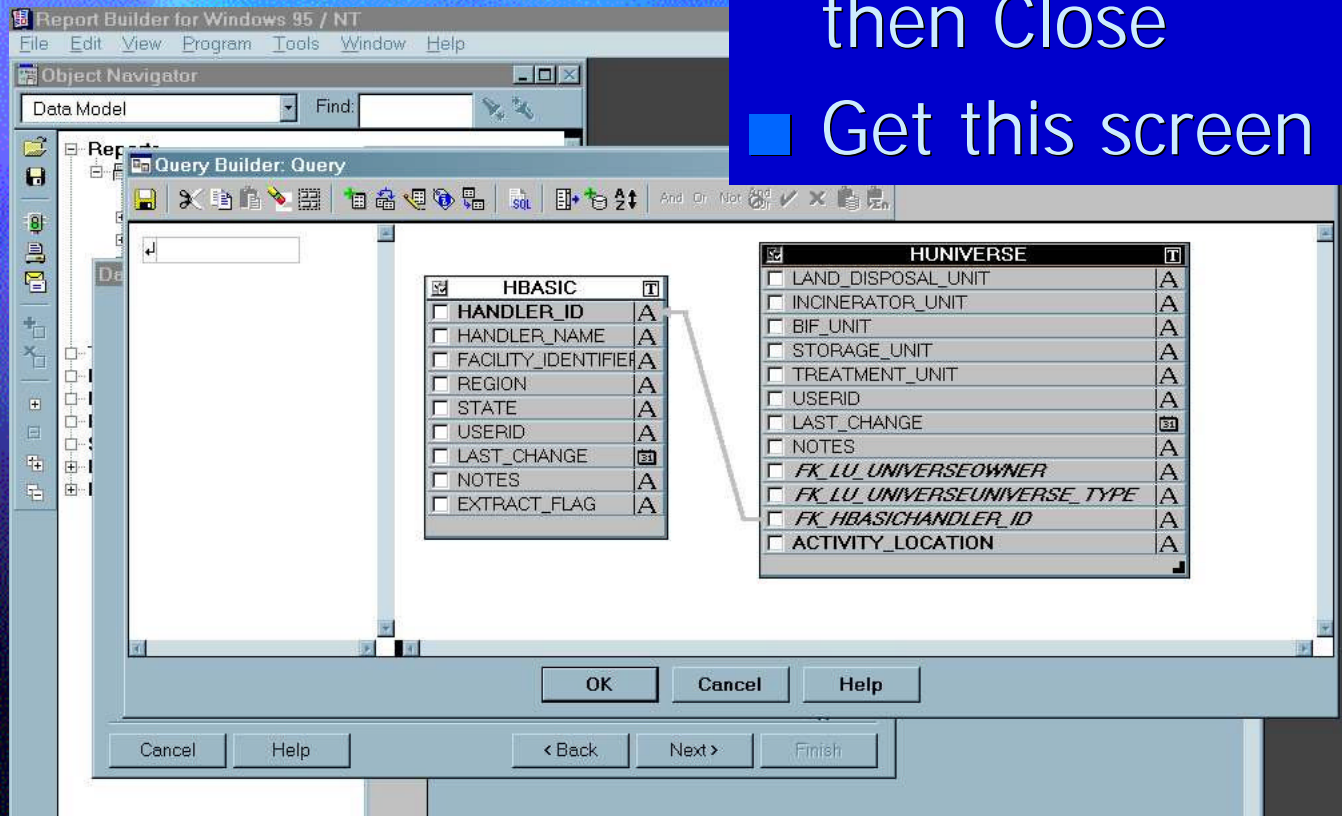
Specifying Data Tables



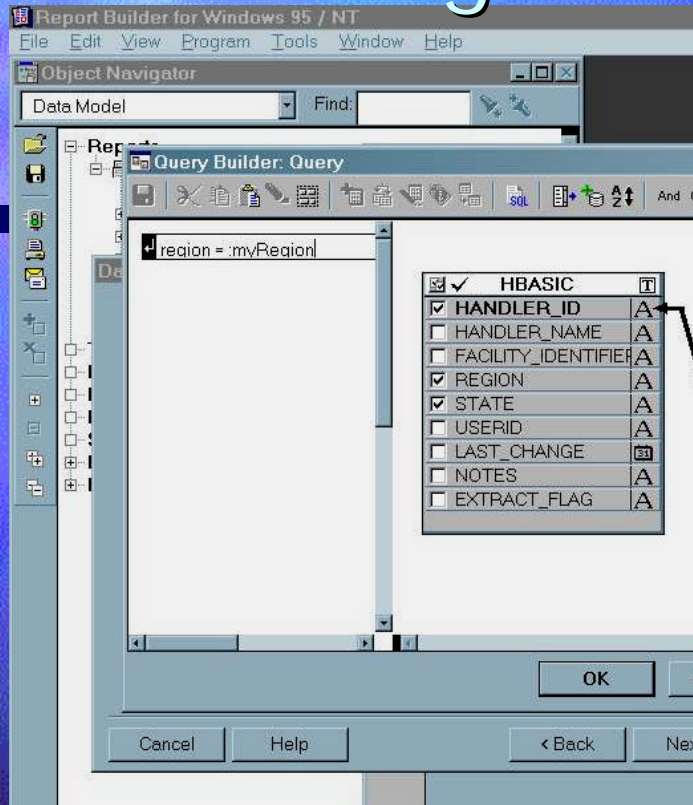
- On the panel *Select Data Tables*, use the drop-down box to get into RCRA (or <LOCAL>, then RCRA); click on Open.
- Scroll down and click once on HBASIC, then click *Include*.

Done with Data Tables

- Now scroll down more to click on HUNIVERSE, then Include, then Close
- Get this screen



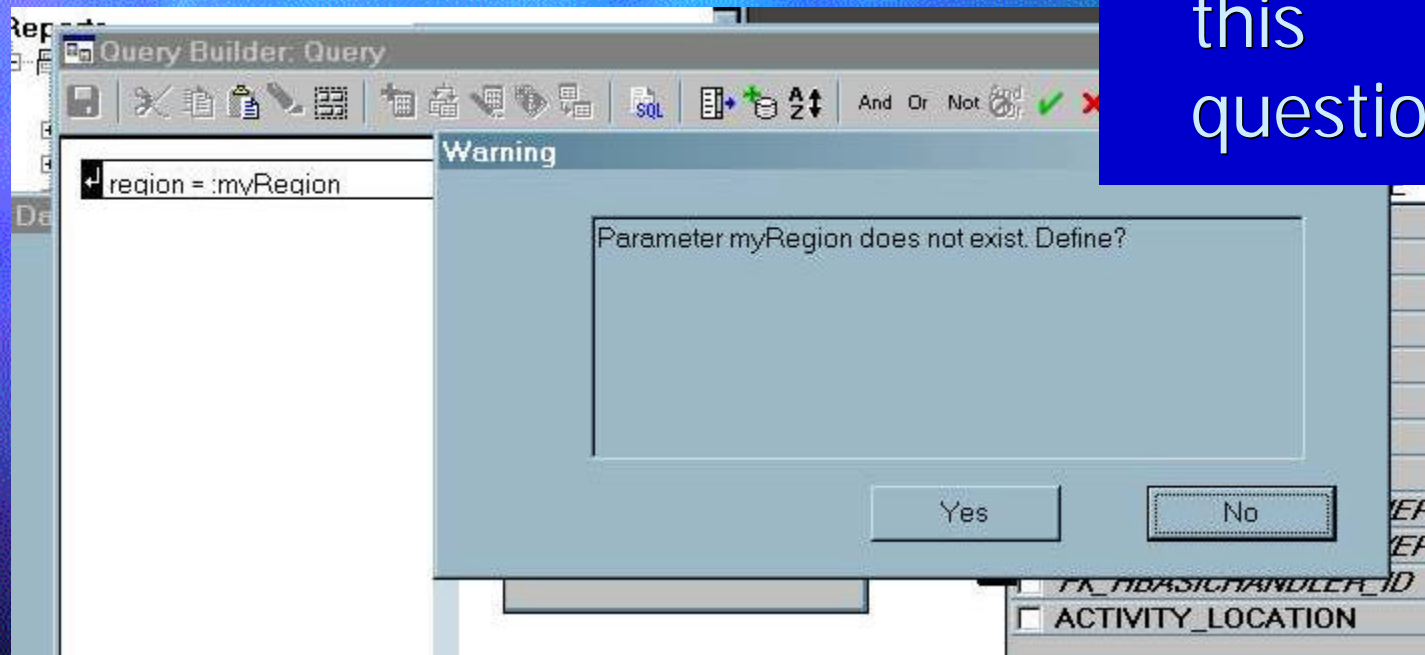
Choosing Columns and Conditions



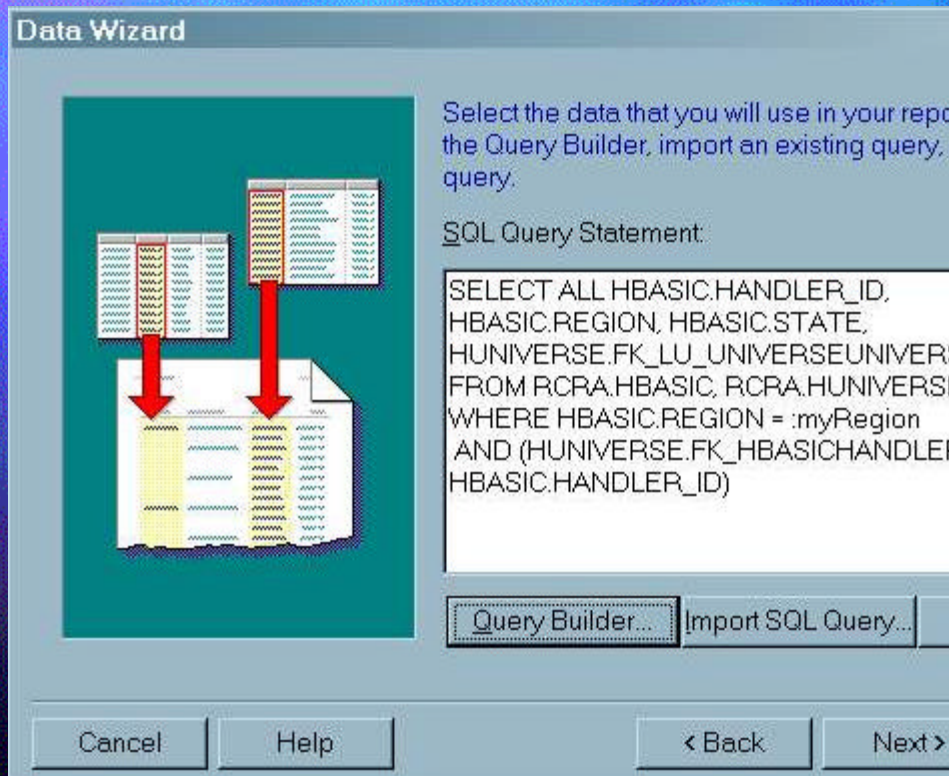
- Note how Query Builder already knows the connection between these two tables.
- Now click in the checkboxes for:
 - Handler_ID
 - Region
 - State, and
 - FK_LU_UniverseUniverse_Type
- Note that the connecting line blackens when we establish the need for the connection.
- Now click in the left-side box, right next to the arrow; these are your WHERE conditions. Since we want to limit this report by Region, type in:
 - region = :myRegion
- The colon (:) in front of the word 'myRegion' tells the program that myRegion will be a user-supplied parameter, not a column in the database.

Define the Parameter

- tap Enter and you get this question:

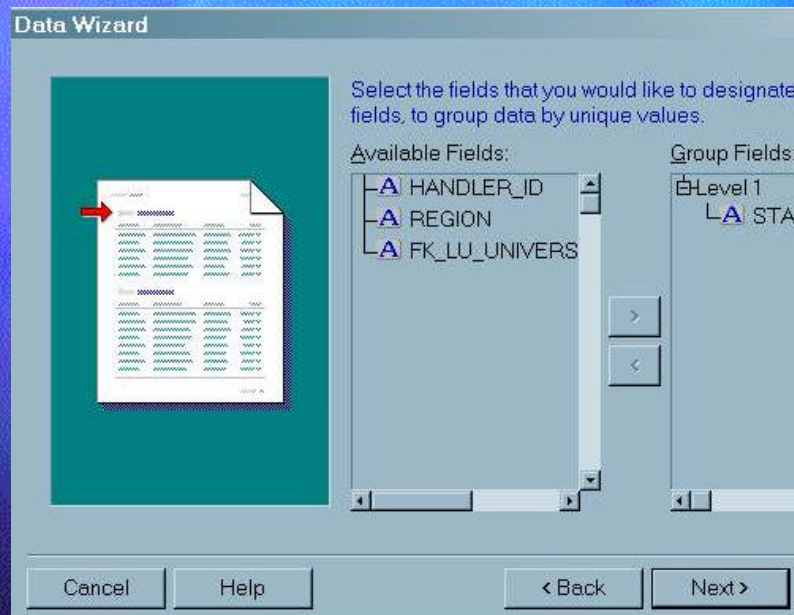


Our Query Is Done



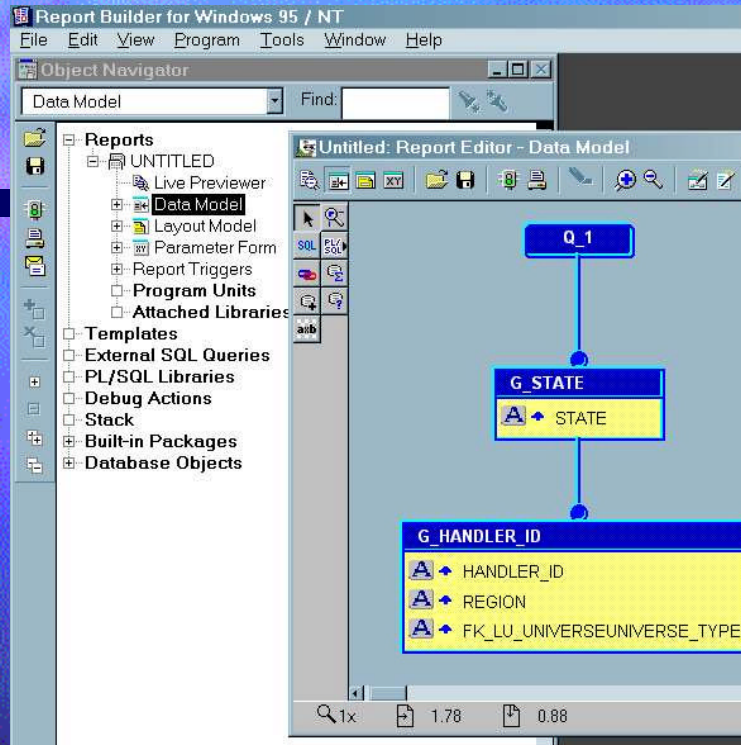
- should we create the parameter? Click Yes.
- now click Okay, and you should get back to this screen.
- Note that the SQL code to accomplish our task now exists in the box, including the parameter myRegion.

Grouping Fields



- click Next, and reconfirm that it's okay to have created the parameter. You should get this screen.
- Click on the field State, then click on the arrow-pointing-right; the field State will move over into the "Group" fields. We do this because we want to count up the universes by State.

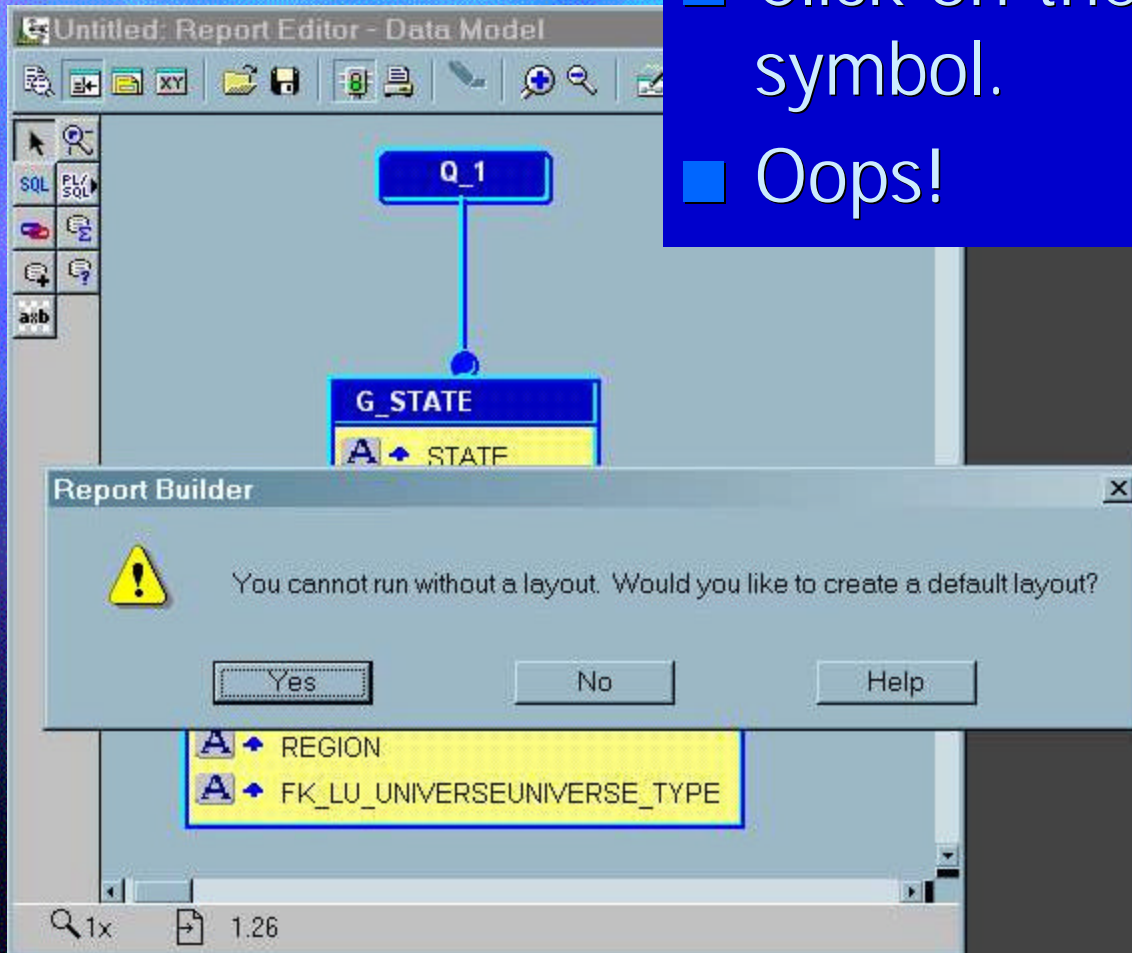
Data Model Chart



- Click *Next*
- Let's *NOT* do totals, since we first want to inspect the raw data.
- Click on *Next* again.
- Click on *Finished* and get this screen.
- So our query has been built, and the output table has been organized into groups, by State, then by Handler information. Note that in this case, we show Region belonging to the Handler, which is okay.

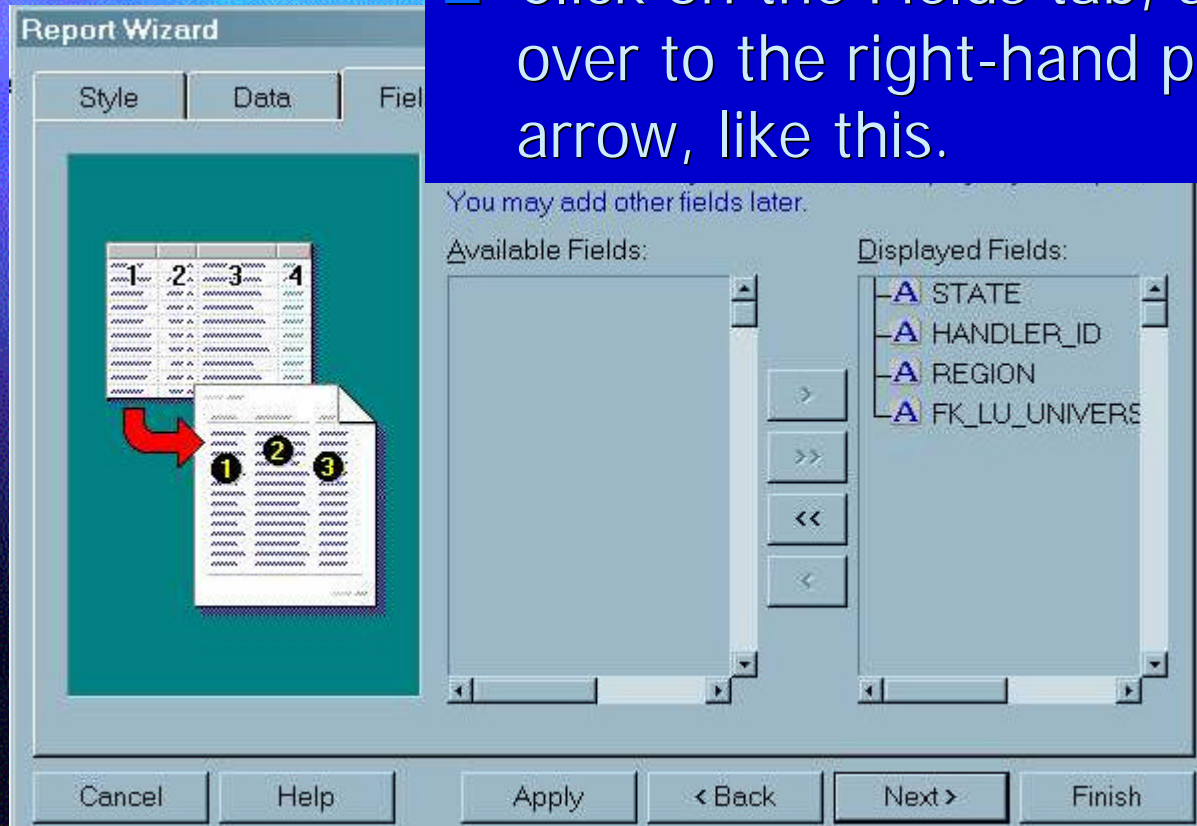
Run it!

- Lets run it!
- Click on the green traffic light symbol.
- Oops!



Making a Default Layout

- That reminds us that we need more than just a *Query*, we need a *Layout*. Say Yes to get a default layout, and get this screen.
- Click on the Fields tab, and move every item over to the right-hand panel using the double arrow, like this.



Still Making a Default Layout

- And then on the Labels tab, reset the Width of Handler_ID to 12.

Report Wizard

Style | Data | Fields | Totals | **Labels** | Template

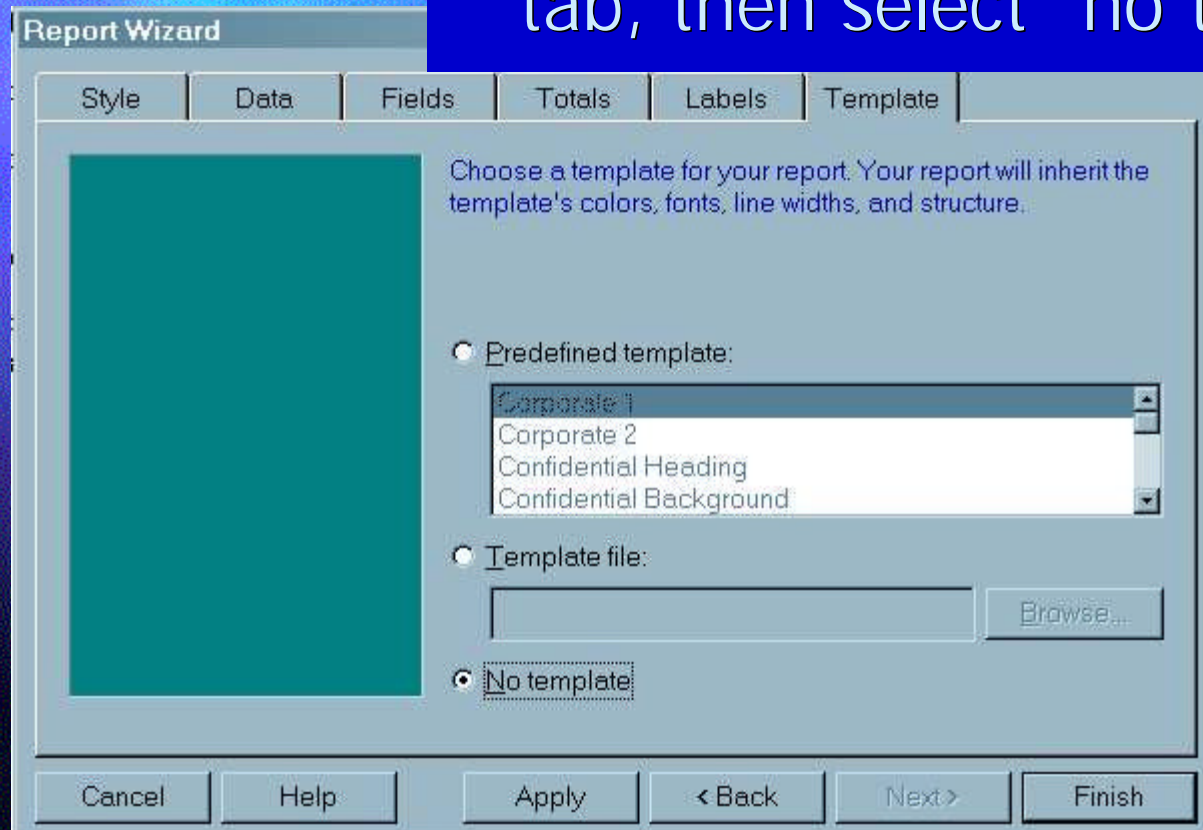
Modify the labels and widths (number of characters) for your fields and totals as desired.

Fields and Totals	Labels	Width
*HANDLER_ID	Handler Id	12
REGION	Region	2
STATE	State	2
FK_LU_UNIVERSEUNIVERSE	Fk Lu Universeuniverse	10

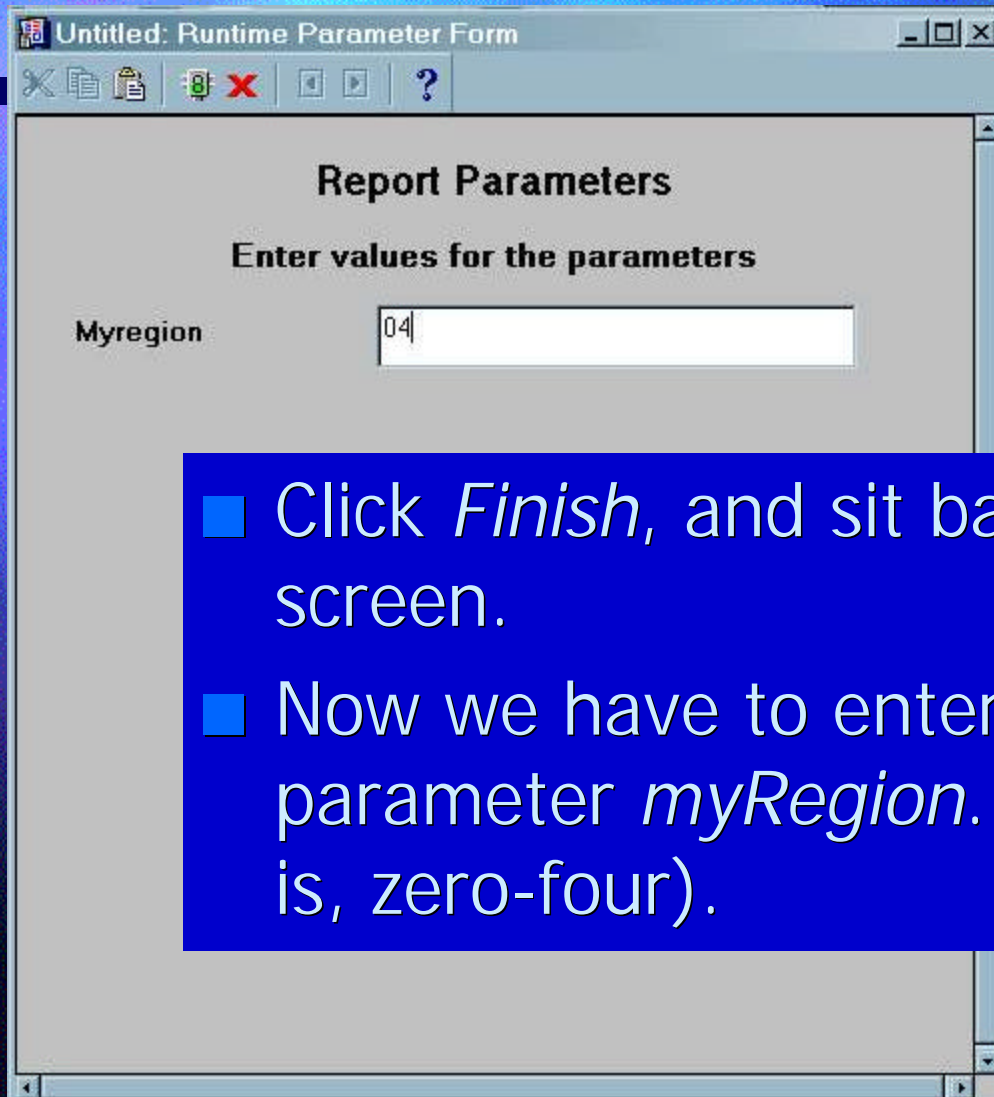
Cancel | Help | Apply | < Back | Next > | Finish

And More Default Layout Stuff

- Since we'll accept a default, but want a simple report, click on the Template tab, then select "no template"



Specify Which Region



Untitled: Runtime Parameter Form

Report Parameters

Enter values for the parameters

Myregion 04

- Click *Finish*, and sit back, and get this screen.
- Now we have to enter a value for our parameter *myRegion*. Type: 04 (that is, zero-four).

We Get Data!

- Again, click on the *traffic light* on the Parameter Form.
- Report Builder tells us: **Formatting Page 1**
- And we get this data!

The screenshot shows the 'Report Builder for Windows 95 / NT' interface. The main window displays a report titled 'Untitled: Report Editor - Live Preview'. The report content is a table with the following columns: State, Handler Id, Region, Fk Lu, and Universeuniverse. The table contains 20 rows of data. The interface includes a menu bar (File, Edit, View, Insert, Format, Arrange, Program, Tools, Window, Help), an Object Navigator on the left, and a toolbar at the top of the report area.

State	Handler Id	Region	Fk Lu	Universeuniverse
AL	AL0000004655	04	CEG	
AL	AL0000004663	04	CEG	
AL	AL0000005702	04	CEG	
AL	AL0000005728	04	CEG	
AL	AL0000005744	04	CEG	
AL	AL0000005819	04	CEG	
AL	AL0000006221	04	CEG	
AL	AL0000006312	04	CEG	
AL	AL0000008359	04	CEG	
AL	AL0000008367	04	CEG	
AL	AL0000008383	04	CEG	
AL	AL0000008409	04	CEG	
AL	AL0000008417	04	CEG	
AL	AL0000017558	04	CEG	
AL	AL0000017608	04	CEG	
AL	AL0000017764	04	CEG	
AL	AL0000018218	04	SQG	
AL	AL0000018234	04	CEG	
AL	AL0000018280	04	CEG	

Examine that Data

- Page into the data using the *Page Down* key.
- Note that soon you find facilities with more than one value for UniverseType.

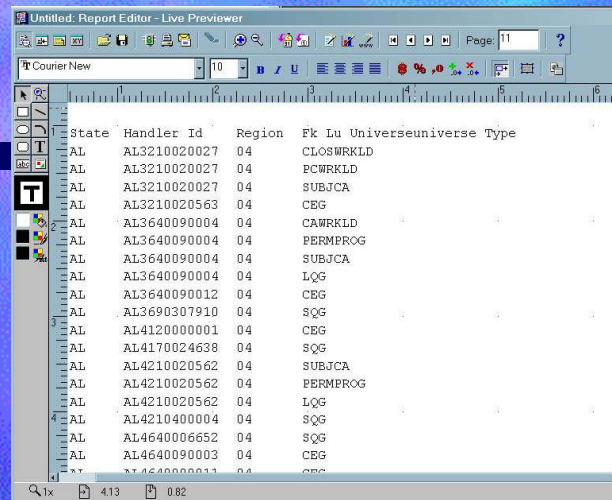
Untitled: Report Editor - Live Previewer

Courier New 10 B I U % .0+ .0+

State	Handler Id	Region	Fk Lu	Universe
AL	AL3210020027	04	CLOSWRKLD	
AL	AL3210020027	04	PCWRKLD	
AL	AL3210020027	04	SUBJCA	
AL	AL3210020563	04	CEG	
AL	AL3640090004	04	CAWRKLD	
AL	AL3640090004	04	PERMPROG	
AL	AL3640090004	04	SUBJCA	
AL	AL3640090004	04	LQG	
AL	AL3640090012	04	CEG	
AL	AL3690307910	04	SQG	
AL	AL4120000001	04	CEG	
AL	AL4170024638	04	SQG	
AL	AL4210020562	04	SUBJCA	
AL	AL4210020562	04	PERMPROG	
AL	AL4210020562	04	LQG	
AL	AL4210400004	04	SQG	
AL	AL4640006652	04	SQG	
AL	AL4640090003	04	CEG	
AL	AL4640000011	04	CEG	

1x 4.13 0.82

Our First Conundrum

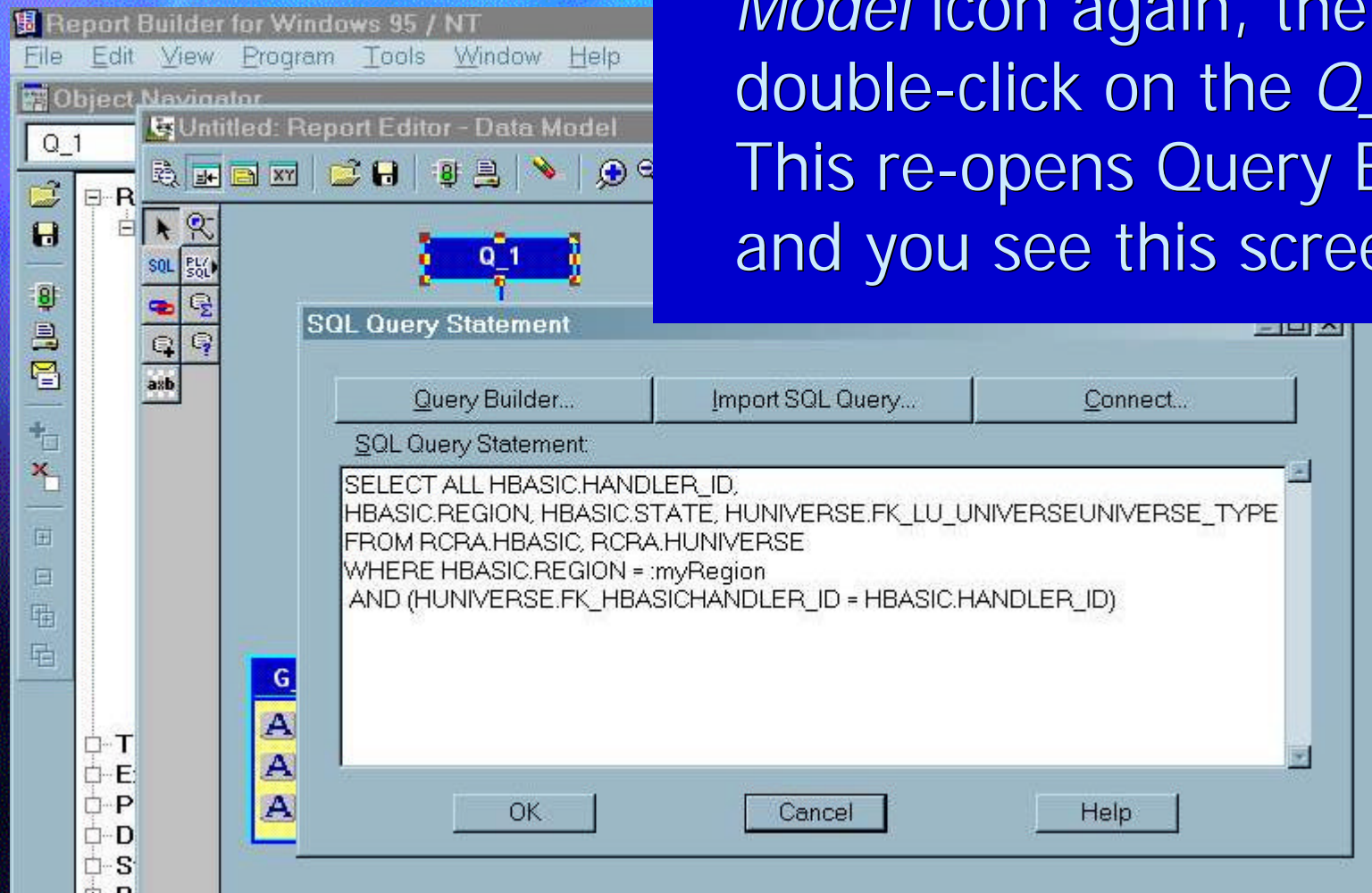


State	Handler Id	Region	Fk Lu Universe	universe Type
AL	AL3210020027	04	CLOSWRKLD	
AL	AL3210020027	04	PCWRKLD	
AL	AL3210020027	04	SUBJCA	
AL	AL3210020563	04	CEG	
AL	AL3640090004	04	CAWRKLD	
AL	AL3640090004	04	PERMPROG	
AL	AL3640090004	04	SUBJCA	
AL	AL3640090004	04	LQG	
AL	AL3640090012	04	CEG	
AL	AL3690307910	04	SQG	
AL	AL4120000001	04	CEG	
AL	AL4170024638	04	SQG	
AL	AL4210020562	04	SUBJCA	
AL	AL4210020562	04	PERMPROG	
AL	AL4210020562	04	LQG	
AL	AL4210400004	04	SQG	
AL	AL4640006652	04	SQG	
AL	AL4640090003	04	CEG	
AL	AL4640090011	04	CEG	

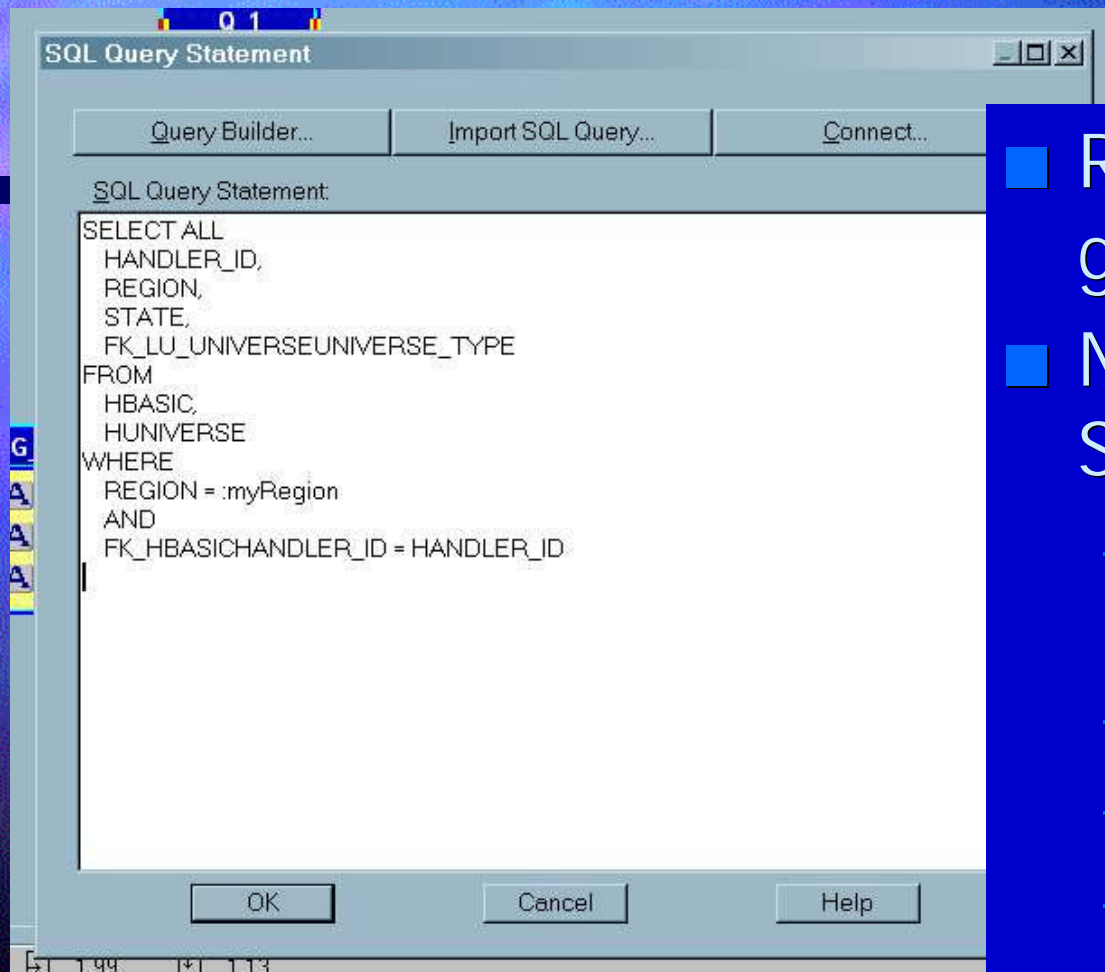
- This means we cannot easily count Universe types. Sure, code like:
 - Select State, count(handler_ID)
 - From HBasic, HUniverse
 - Where ...universeType = 'LQG'...
- counts the LQGs, but the same report cannot also count SQGs, etc.
- However, if we could convert these universeTypes into numbers (like 1 or 0, based on what text they contain), we could SUM the whole state and that would also be a count of that universeType.
- And if our output table had several columns of universeType, we could examine each column with a different basis for converting the text into 1's and 0's. Since this is a post-output table manipulation, we should use DECODE().
- So lets edit some SQL manually!

Re-enter Query Builder

- Close this *Live Previewer*.
- Double-click on the *Data Model* icon again, then double-click on the *Q_1* box. This re-opens Query Builder, and you see this screen.



Clean Up The SQL



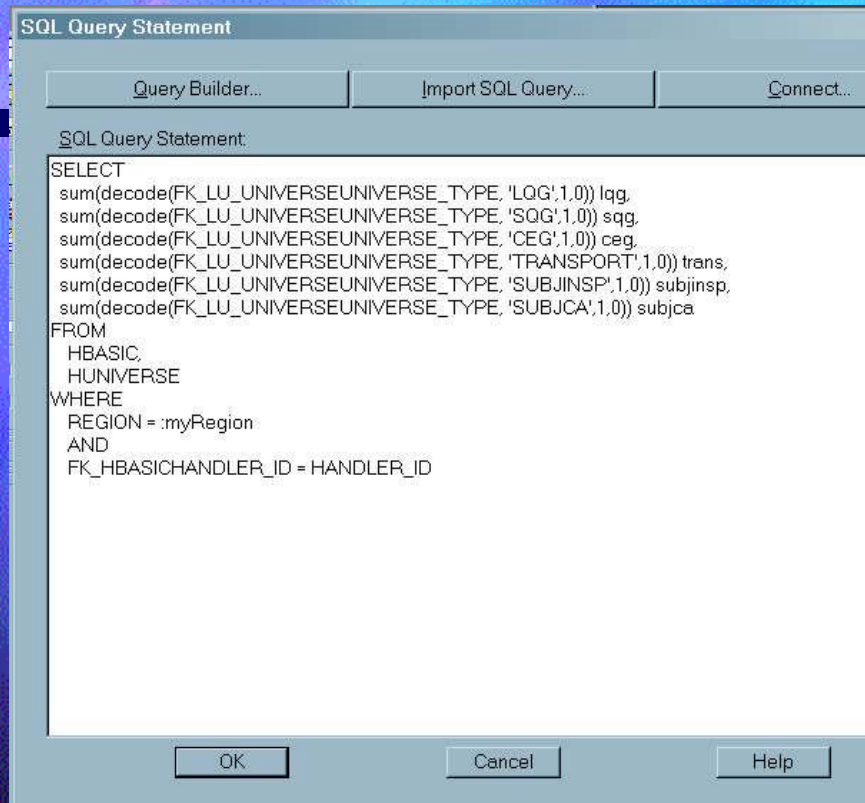
- Resize the window to give us more room.
- Now let's clean up the SQL already there.
 - Eliminate the parentheses.
 - Add Line breaks.
 - Add Indents.
 - Drop Table names, since column names are unique.
- Get this screen.

Test Our Clean Up

- We should rerun the code at this point to check that we haven't screwed anything up.
- So click *Okay*, then the *traffic light*, then '04', then the *light* again, and you should get this screen.
- Yes, it's the same data. And notice we're on the same page we last viewed last time.
- Now lets add our DECODE in.

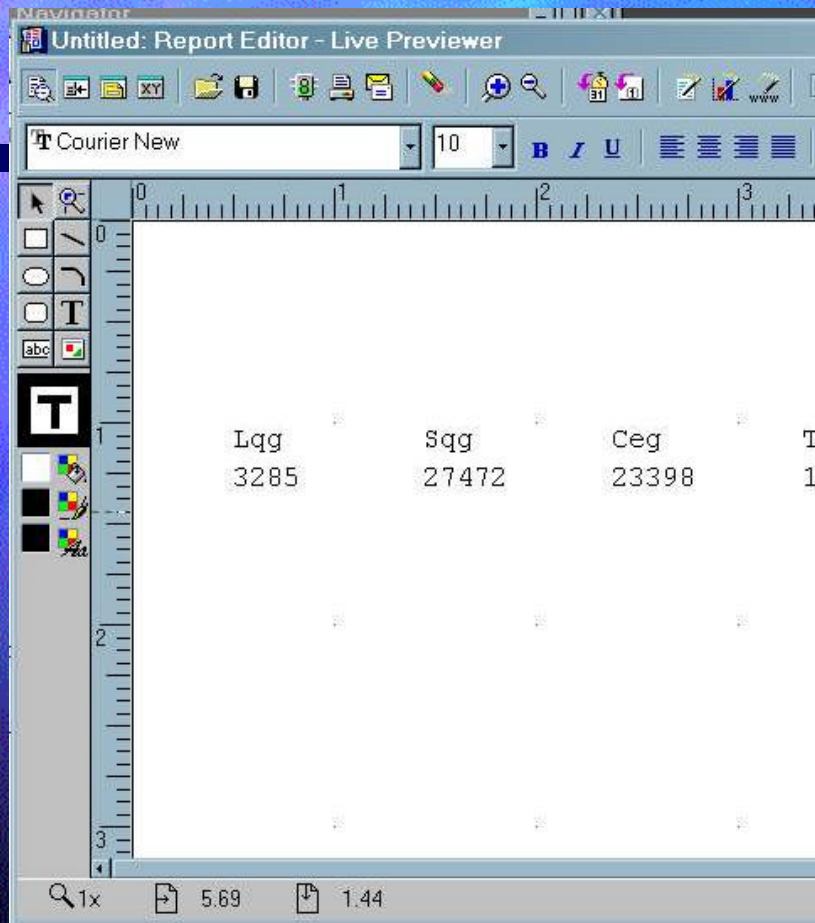
State	Handler Id	Region	Fk Lu	Universe	universe T
AL	AL3210020027	04	CLOS	WRKLD	
AL	AL3210020027	04	PC	WRKLD	
AL	AL3210020027	04	SUBJ	CA	
AL	AL3210020563	04	CE	G	
AL	AL3640090004	04	CA	WRKLD	
AL	AL3640090004	04	PER	M	PROG
AL	AL3640090004	04	SUBJ	CA	
AL	AL3640090004	04	LQ	G	
AL	AL3640090012	04	CE	G	
AL	AL3690307910	04	SQ	G	
AL	AL4120000001	04	CE	G	
AL	AL4170024638	04	SQ	G	
AL	AL4210020562	04	SUBJ	CA	
AL	AL4210020562	04	PER	M	PROG

Add In Sum(Decode())



- First, return to the first page.
- Close the Live Previewer,
- Open the Data Model,
- Double-click on Q_1 and resize it.
 - We can drop the word ALL.
 - We don't need Handler_ID, Region, or State columns anymore.
 - Edit in: **sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'LQG',1,0)) lqq**
- Notes:
 - This will count all the folks who have universe type = LQG.
 - That last 'lqq' is an alias column name.
- Now edit copy and paste this in five more times and make those fit:
- SQG, CEG, TRANSPORT, SUBJINSP, SUBJCA
- Watch your commas!

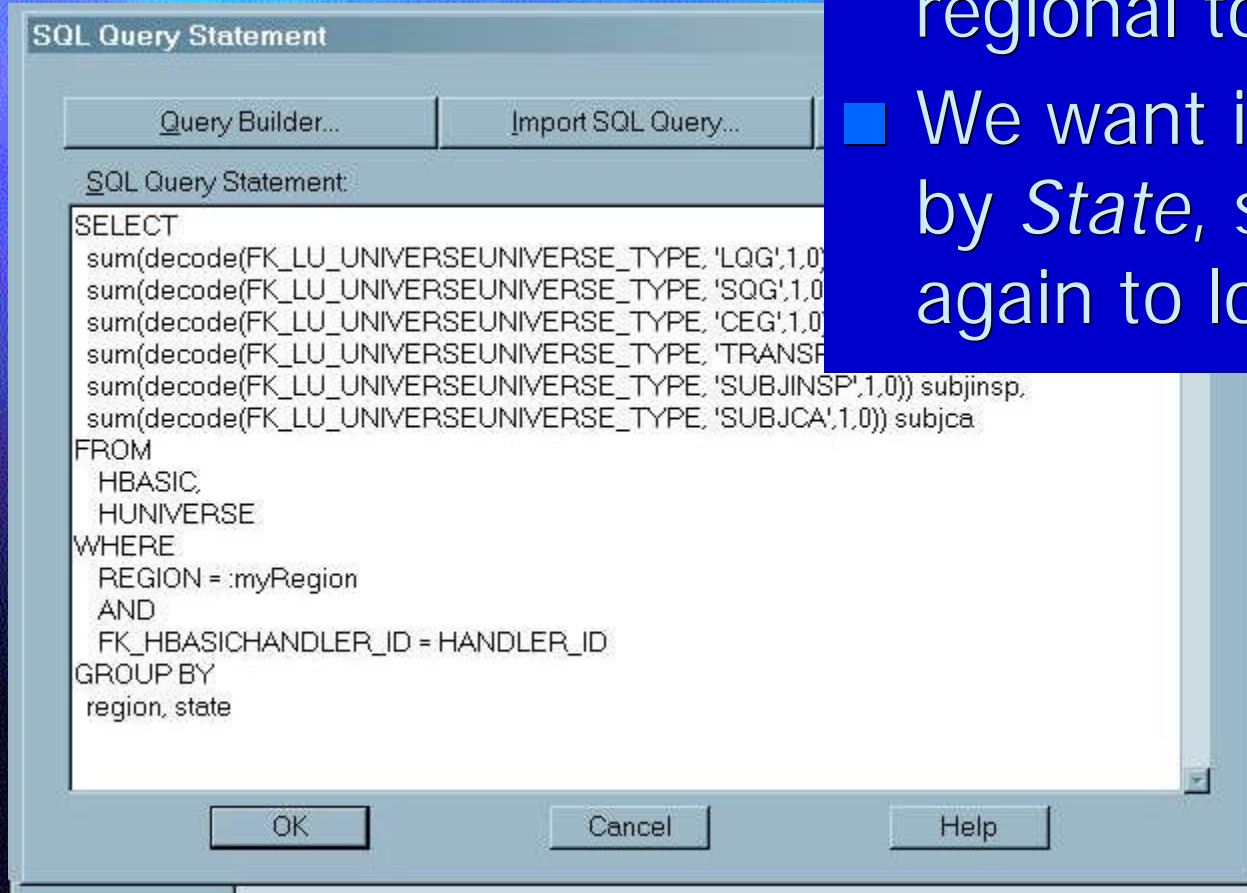
Recreate a Default Layout



- Click *Okay*, run it.
- Oops! It doesn't work!
- X-boxes mean our Layout no longer matches our query (of course), so recreate a default layout via Report Wizard (see above).
- Display all fields, do NO totals here.
- When you finish, it runs and looks like this screen.

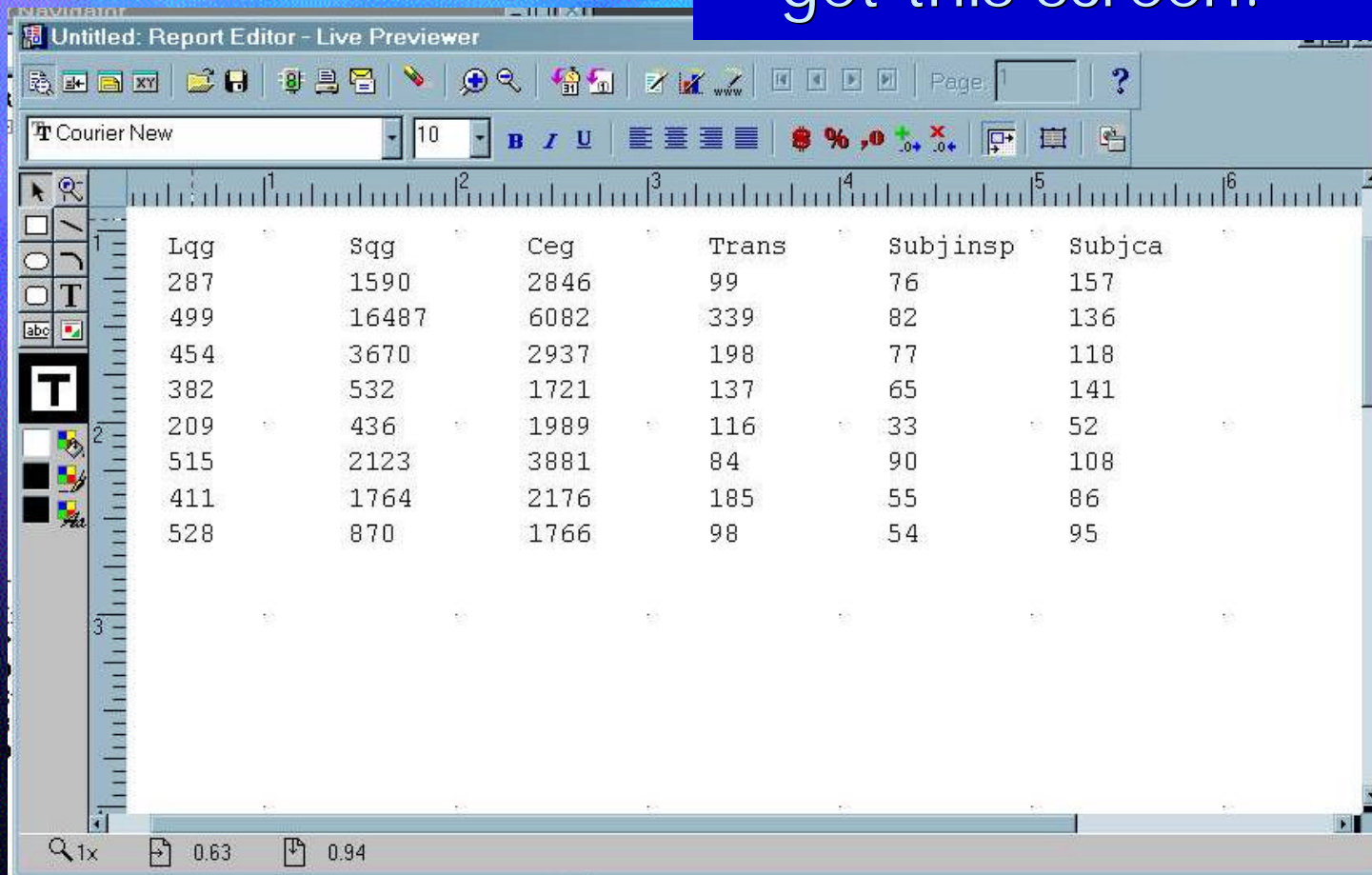
Remember to 'Group By'

- Oops again! Sum() needs a "group by" if we want anything better than regional totals!
- We want it by *Region* and by *State*, so edit the code again to look like this:



It Works!

- Now run it. You should get this screen.



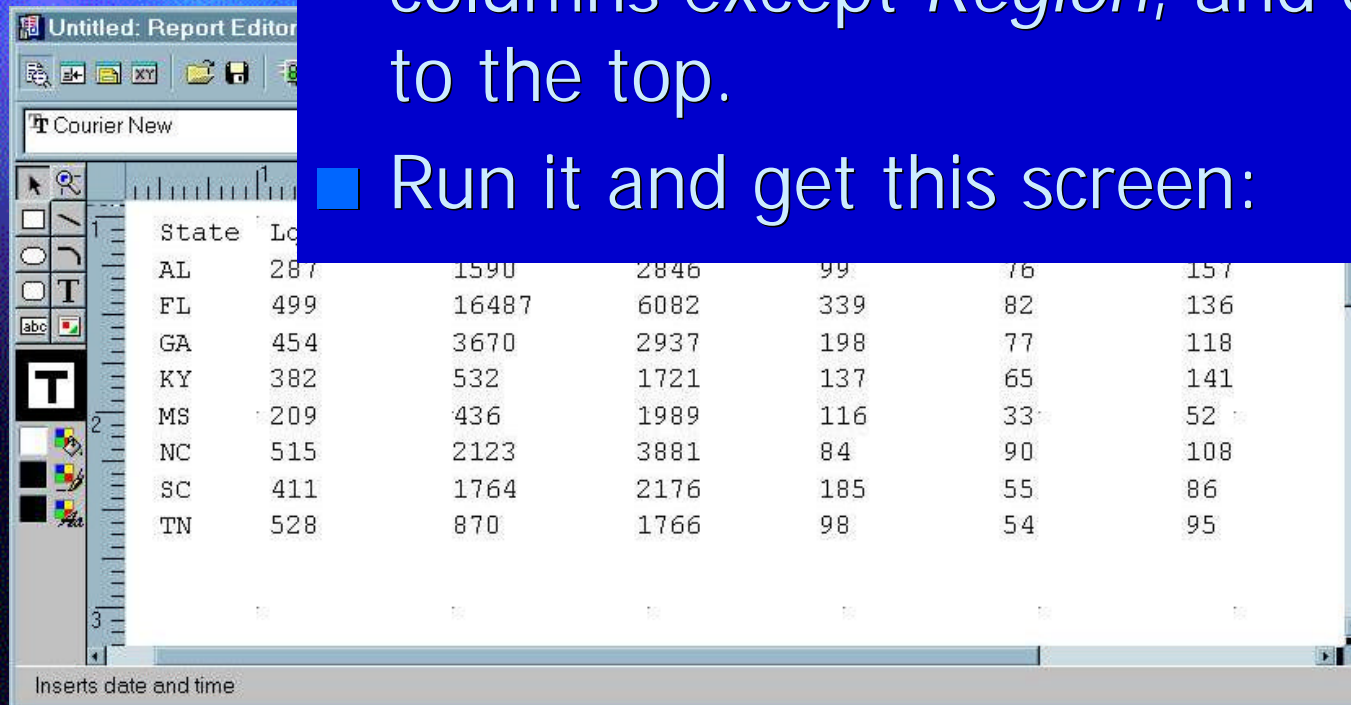
The screenshot shows a software window titled 'Untitled: Report Editor - Live Previewer'. The window has a menu bar, a toolbar with various icons, and a status bar at the bottom. The main area displays a table with 6 columns and 8 rows of data. The columns are labeled 'Lgg', 'Sgg', 'Ceg', 'Trans', 'Subjinsp', and 'Subjca'. The data is as follows:

Lgg	Sgg	Ceg	Trans	Subjinsp	Subjca
287	1590	2846	99	76	157
499	16487	6082	339	82	136
454	3670	2937	198	77	118
382	532	1721	137	65	141
209	436	1989	116	33	52
515	2123	3881	84	90	108
411	1764	2176	185	55	86
528	870	1766	98	54	95

The status bar at the bottom shows '1x' magnification, '0.63' and '0.94' values, and a 'Page: 1' indicator.

Now Identify States

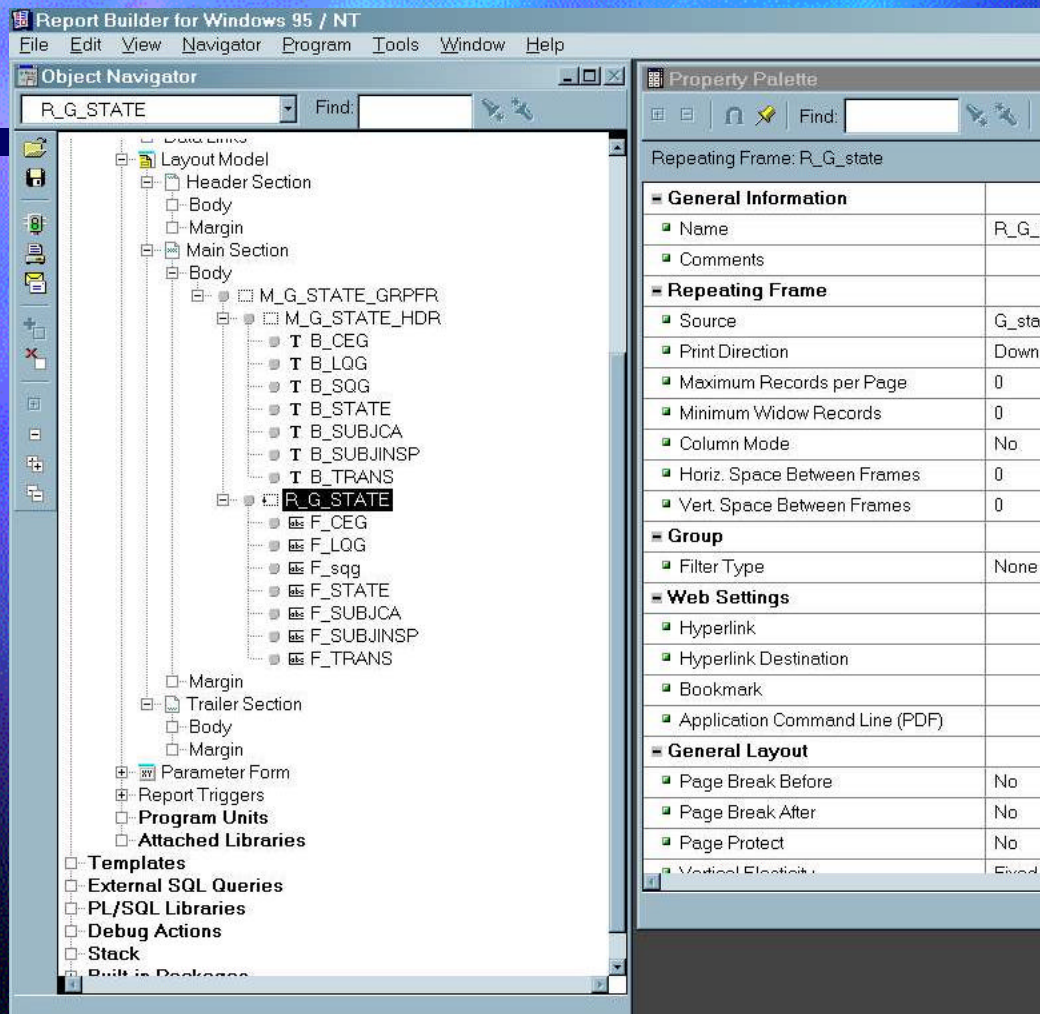
- Great! But we need to know the States too.
- Edit the SQL in *Report Wizard* this time, adding *State* and *Region* back into the *Select* statement, and then add a *State* field (not a *Region* one yet) by displaying all columns except *Region*, and dragging *State* to the top.
- Run it and get this screen:



The screenshot shows a report editor window titled 'Untitled: Report Editor'. The report content is a table with 7 columns and 8 rows of data. The first column is labeled 'State' and the second is labeled 'Lo'. The data rows correspond to the states AL, FL, GA, KY, MS, NC, SC, and TN. The table contains numerical values for each state across the remaining five columns.

State	Lo					
AL	287	1590	2846	99	76	157
FL	499	16487	6082	339	82	136
GA	454	3670	2937	198	77	118
KY	382	532	1721	137	65	141
MS	209	436	1989	116	33	52
NC	515	2123	3881	84	90	108
SC	411	1764	2176	185	55	86
TN	528	870	1766	98	54	95

Explore the Object Navigator



- This is a non-hierarchical universe count report for any Region in the nation.
- Now close the *Previewer*, and let's examine the *Object Navigator*. Expand the *Layout Model* and notice the nested-frame structure of this report.

Navigator Connects to Layout

- Note that the repeating frame *R_G_State* is sourced to the group *G_State*, and up in the *Data Model*, that group has the same set of child columns, matching these fields.
- Now keep the *Navigator* and *Property palette* open, and open *Layout Model*. By clicking on different items in the *Navigator*, we select different items in the *Layout*, and vice versa.

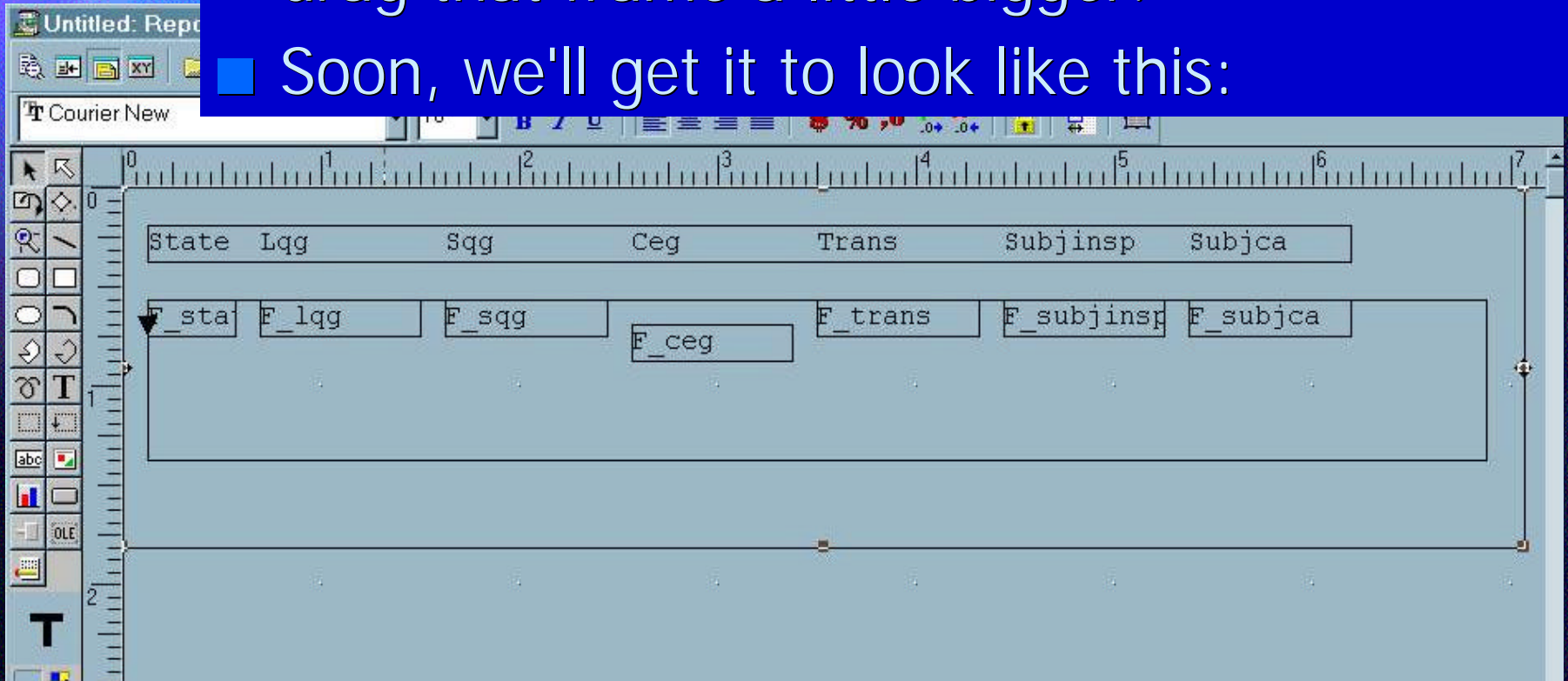
The screenshot displays the Report Builder application with three main components visible:

- Object Navigator:** A tree view on the left showing the report structure. The 'Layout' folder is expanded, revealing sections like 'Header Section', 'Main Section', and 'Trailer Section'. Under 'Main Section' > 'Body', the 'R_G_STATE' frame is selected, which contains a repeating frame 'R_G_STATE' and several child frames like 'F_CEG', 'F_LQG', 'F_SQG', 'F_STATE', 'F_SUBJCA', 'F_SUBJINSP', and 'F_TRANS'.
- General Information:** A table on the right showing properties for the selected item 'F_sqg'.

General Information	
Name	F_sqg
Comments	
Field	
Source	sqg
- Layout Model:** A window titled 'Untitled: Report Editor - Layout Model' showing a visual representation of the report layout. It features a ruler at the top and a grid of frames. The frames are labeled with their names: 'State', 'Lqg', 'Sqg', 'Ceg', 'Trans', and 'Subjin'. Below these, the corresponding field names are listed: 'F_sta', 'F_lqg', 'F_sqg', 'F_ceg', 'F_trans', and 'F_subjin'.

Expanding Frames

- Now let's expand this layout so we can insert a field for *Region*.
- Select any field, then click on the *Parent-Frame* button until we're outer-most, then drag that frame a little bigger.
- Soon, we'll get it to look like this:



Adding Region to the Report

- Add a label using the **T** (text) tool, type into it the word-and-colon *Region:*
- Justify it right.
- Add a field using the **abc** tool, and set its source property to be *myRegion* in the dropdown box.

The screenshot shows a report designer interface with a table. The table has columns: State, Lqg, Sqg, Ceg, Trans, Subjinsp, and Subjca. A 'Region:' label is positioned above the 'Sqm' column, and a dropdown menu is open showing '04'. The table contains data for two states: AL and FL.

State	Lqg	Sqm	Ceg	Trans	Subjinsp	Subjca
AL	287	1590	2846	99	76	157
FL	499	16487	6082	339	82	136

Making It Pretty

■ Now let's:

- remove the lines around the Region field (select the field, click on Line color, select *No Line*)
- left justify it
- add colors to state text
- set the fill color in the title frame and the field frame
- make the region and states bold
- right justify the labels and fields all at once by selecting them while holding down the shift key
- close up the frames

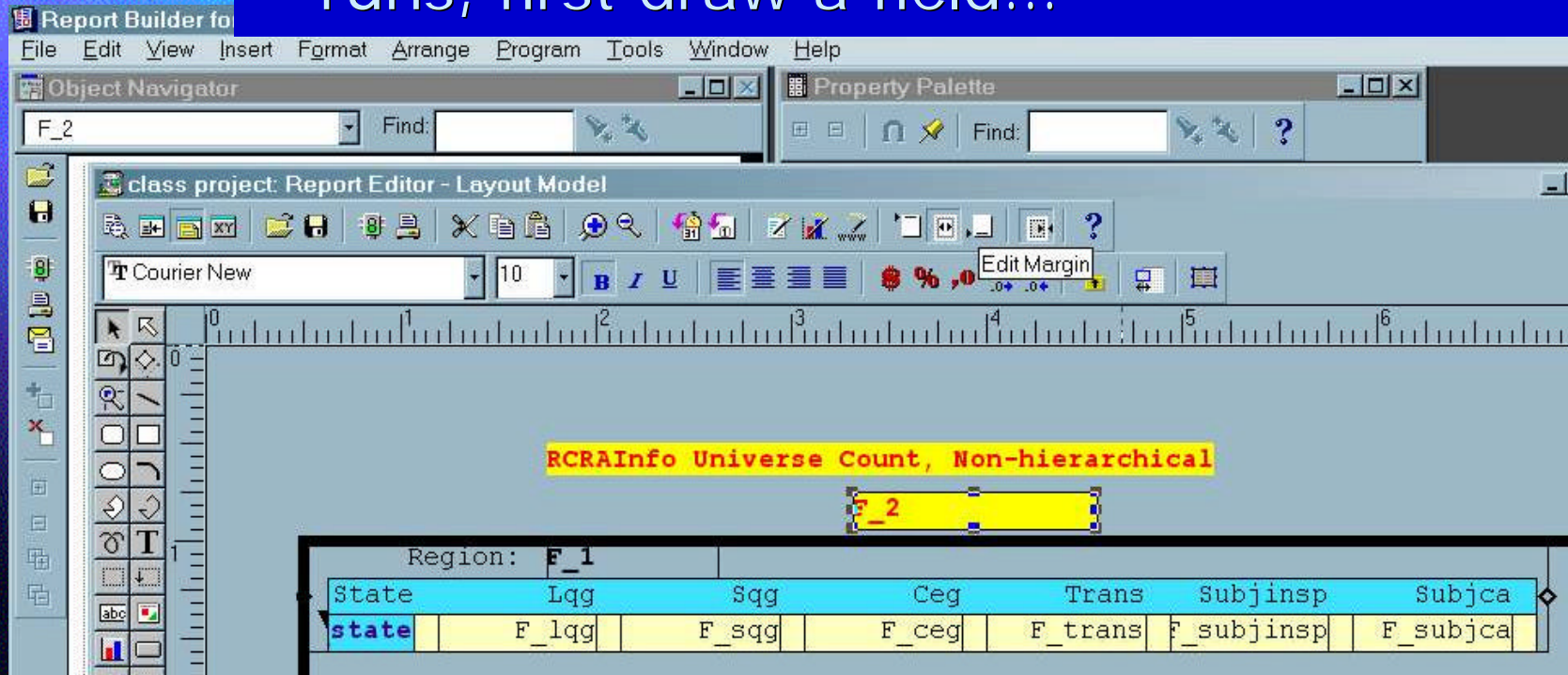
■ And run it. You should get this screen.

Region: **04**

State	Lqg	Sqg	Ceg	Trans	Subjinsp	Subjca
AL	287	1590	2846	99	76	157
FL	499	16487	6082	339	82	136
GA	454	3670	2937	197	77	118
KY	382	532	1721	137	65	141
MS	209	436	1989	116	33	52
NC	515	2123	3881	84	90	108
SC	411	1764	2176	185	55	86
TN	528	870	1766	98	54	95

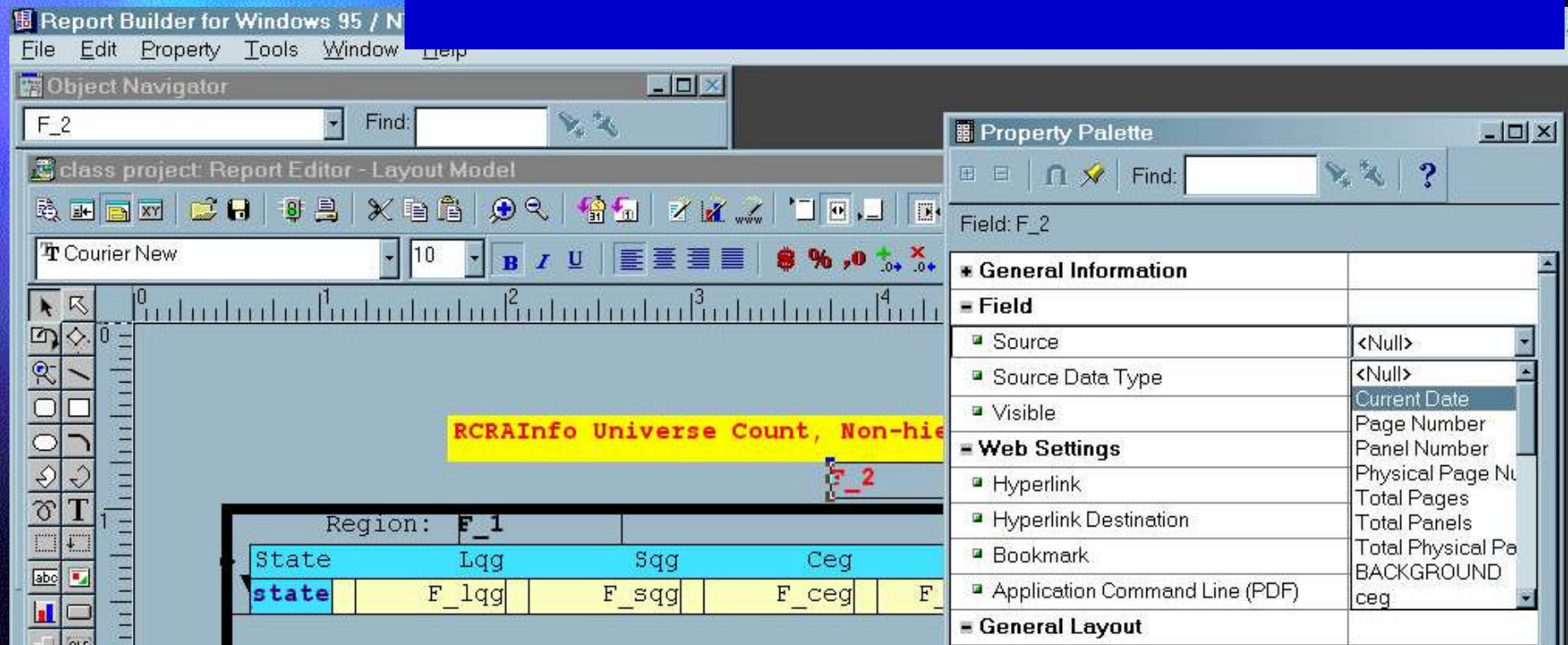
Add a Title and Date

- click on the Edit Margin button
- draw a Text box, type in a title and color it.
- Now, to add the date on which the report runs, first draw a field...



Automate the Date

- ...open that field's *Property Palette*
- Click *Source*, use the *dropdown* to select *Current Date*.



A Finished Product

- Add another label "Report run on:"
- Adjust justifications, etc.
- Run it, and get this: our finished report!

RCRAInfo Universe Count, Non-hierarchical

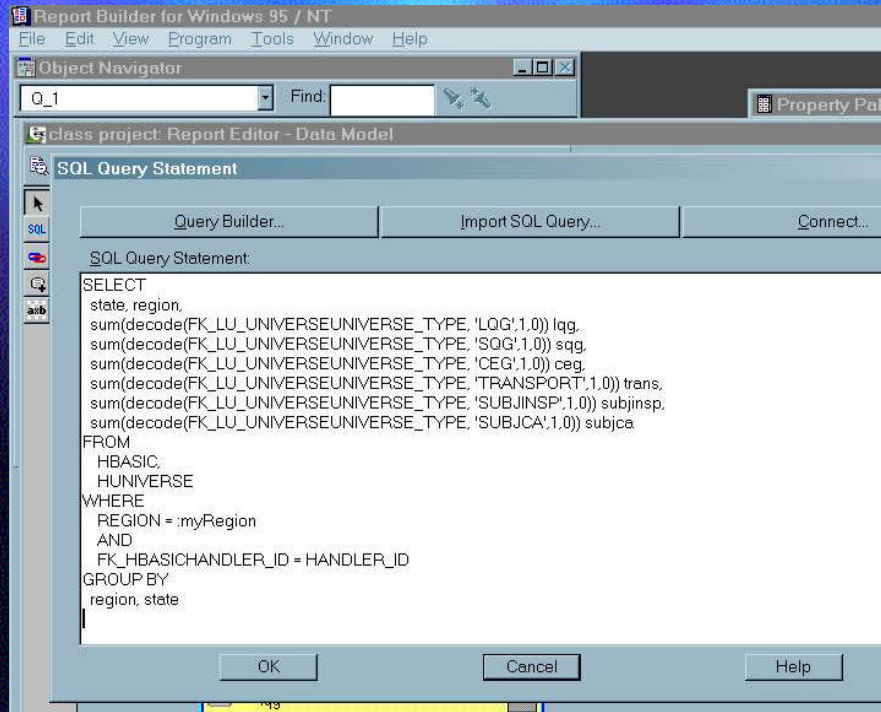
Report run on: 13-SEP-01

Region: 04

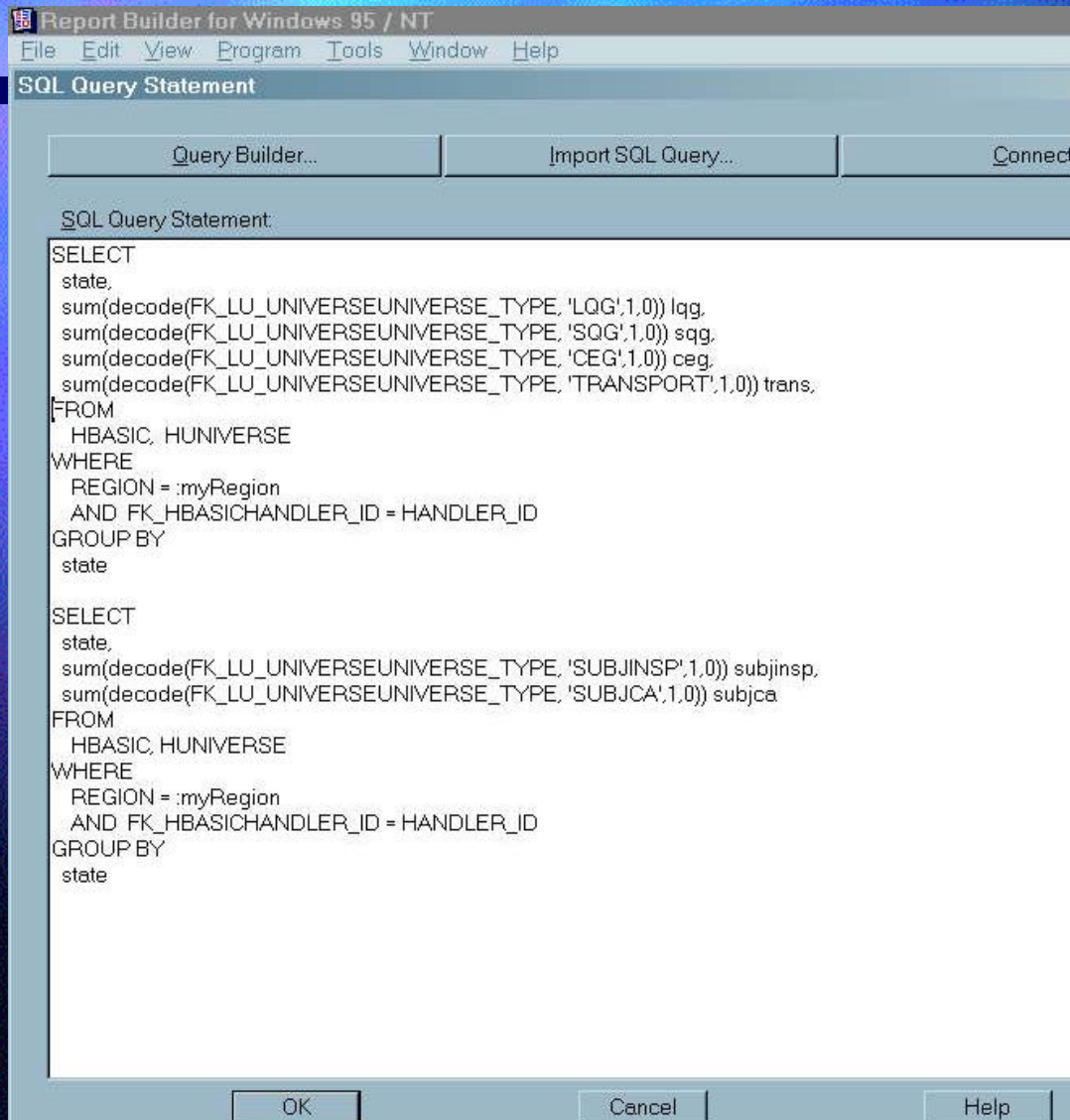
State	Lqg	Sqg	Ceg	Trans	Subjinsp	Subjca
AL	287	1590	2846	99	76	157
FL	499	16487	6082	339	82	136
GA	454	3670	2937	197	77	118
KY	382	532	1721	137	65	141
MS	209	436	1989	116	33	52
NC	515	2123	3881	84	90	108
SC	411	1764	2176	185	55	86
TN	528	870	1766	98	54	95

What about Hierarchy?

- Now consider the **Hierarchical** report.
- The difference is that for Generators and Transporters, we should not count them if they are **also** Subject to Inspection (*i.e.*, if they have the **UniverseType** = **SUBJINSP**).
- We can do this by splitting our current query into two parts (one counting Gens and Trans, the other counting everything else), and condition only the first part to NOT count those who are SUBJINSP.
- To split the query, first open the Data Model, double-click on Q_1, and enlarge the edit box.



Duplicate and Edit SQL



- Now make two separate Select statements by copying the whole thing, pasting it in and deleting certain lines. Also, since the *region* field in our report is sourced to the *parameter* and not the *column*, we don't need to retrieve the region column any more.

Merge Them Into One Select

SQL Query Statement

```
SELECT state, lqg, sqg, ceg, trans, subjinsp, subjca
FROM
(SELECT
  state,
  sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'LQG',1,0)) lqg,
  sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SQG',1,0)) sqg,
  sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'CEG',1,0)) ceg,
  sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'TRANSPORT',1,0)) trans,
FROM
  HBASIC, HUNIVERSE
WHERE
  REGION = :myRegion
  AND FK_HBASICHANDLER_ID = HANDLER_ID
GROUP BY
  state) ,
(SELECT
  state secondState,
  sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SUBJINSP',1,0)) subjinsp,
  sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SUBJCA',1,0)) subjca
FROM
  HBASIC, HUNIVERSE
WHERE
  REGION = :myRegion
  AND FK_HBASICHANDLER_ID = HANDLER_ID
GROUP BY
  state)
WHERE
  state = secondState
```

- Now put both of those SELECTS in parentheses; we will use THEM as our source tables in a new SELECT that gets columns with the same name as these aliases.
- Since having duplicate column names makes things difficult, give the second SELECT's state column a new alias of 'secondState'. Now our WHERE that links these two sub-queries links them by state.

Exclude All SUBJINSP IDs

- Run it to confirm we have not really changed anything. And watch your commas!
- Good, it works. Now add a condition to the first sub-query's WHERE clause:
 - ...and handler_id not in (select FK_HBASICHANDLER_ID from huniverse where FK_LU_UNIVERSEUNIVERSE_TYPE = 'SUBJINSP')

```
SQL C
SELECT
FROM
(SELECT
state,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'LQG',1,0)) lqg,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SQG',1,0)) sqg,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'CEG',1,0)) ceg,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'TRANSPORT',1,0)) trans
FROM
HBASIC, HUNIVERSE
WHERE
REGION = :myRegion
AND FK_HBASICHANDLER_ID = HANDLER_ID
and handler_id not in ( select FK_HBASICHANDLER_ID from huniverse where FK_LU_UNIVERSEUNIVERSE_TYPE = 'SUBJINSP' )
GROUP BY
state) ,
(SELECT
state secondState,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SUBJINSP',1,0)) subjinsp,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SUBJCA',1,0)) subjca
FROM
HBASIC, HUNIVERSE
WHERE
REGION = :myRegion
AND FK_HBASICHANDLER_ID = HANDLER_ID
GROUP BY
state)
WHERE
state = secondState
```


Run It Again

- Run it again, and get this screen. Yes, the new numbers are correct...

RCRAInfo Universe Count, Non-hierarchical

Report run on: 13-SEP-01

Region: 04

State	Lqg	Sqg	Ceg	Trans	Subjinsp	Subjca
AL	245	1578	2832	91	76	157
FL	451	16473	6074	318	82	136
GA	401	3665	2931	180	77	118
KY	340	531	1716	125	65	141
MS	184	434	1987	111	33	52
NC	457	2111	3871	67	90	108
SC	364	1760	2172	180	55	86
TN	493	865	1764	90	54	95

But That Title...

- ...but the title is wrong! Let's add a new label (instead of revising the old one).
- This is your screen now.

RCRAInfo Universe Count, Hierarchical

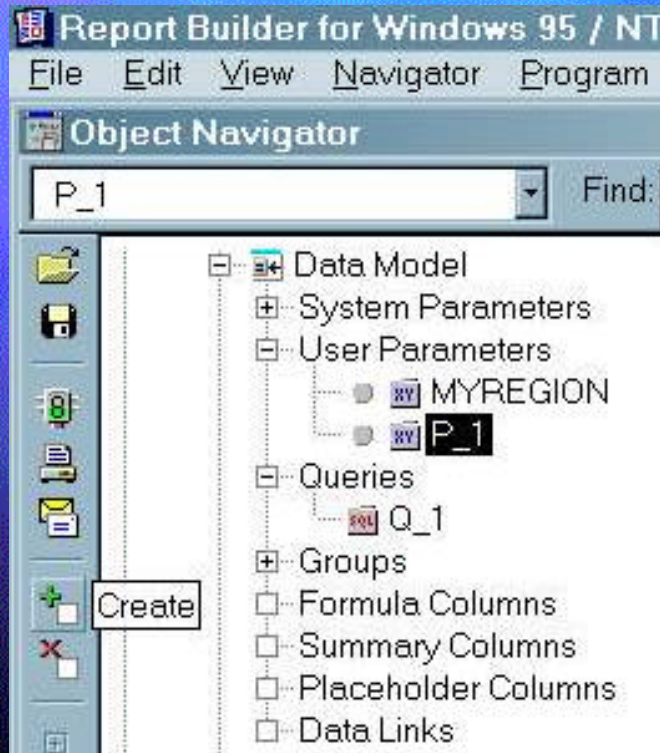
RCRAInfo Universe Count, Non-hierarchical

Report run on: F_2

Region: F_1

State	Lqg	Sqg	Ceg	Trans	Subjinsp	Subjca
state	F_lqg	F_sqg	F_ceg	F_trans	F_subjinsp	F_subjca

But Two Titles?



- Why don't we let the user choose which kind of count they want, then
 - show only the appropriate title, and
 - run only the appropriate SQL!
- We'll need a parameter to do that, and call it: *yesHier*
- We create this new parameter in the *Object Navigator*, expanding the *Data Model* and the *User Parameters* nodes, then click on the "Plus box" in the toolbar.

Modifying the Parameter

- Change its name by clicking on P_1 in the *Navigator*, typing in *yesHier*, tap Enter.
- Then give it an *Initial Value* of 1 in the *Parameters Palette* (we'll say 1 = *do a hier count*, anything else = *do a non-hier count*)

The screenshot displays the Report Builder application interface. On the left, the **Object Navigator** pane shows a tree structure of report components. Under the **Data Model** section, the **User Parameters** folder is expanded, showing two parameters: **MYREGION** and **YESHIER**. The **YESHIER** parameter is selected. The main workspace shows a menu bar with **File**, **Edit**, and **Properties**. On the right, the **Property Palette** pane is open, displaying the configuration for the selected **User Parameter: yesHier**. The palette includes a **General Information** section and a **Parameter** section with the following settings:

* General Information	
Parameter	
Datatype	Number
Width	20
Input Mask	
Initial Value	1
Validation Trigger	
List of Values	

Modifying the Parameter Form

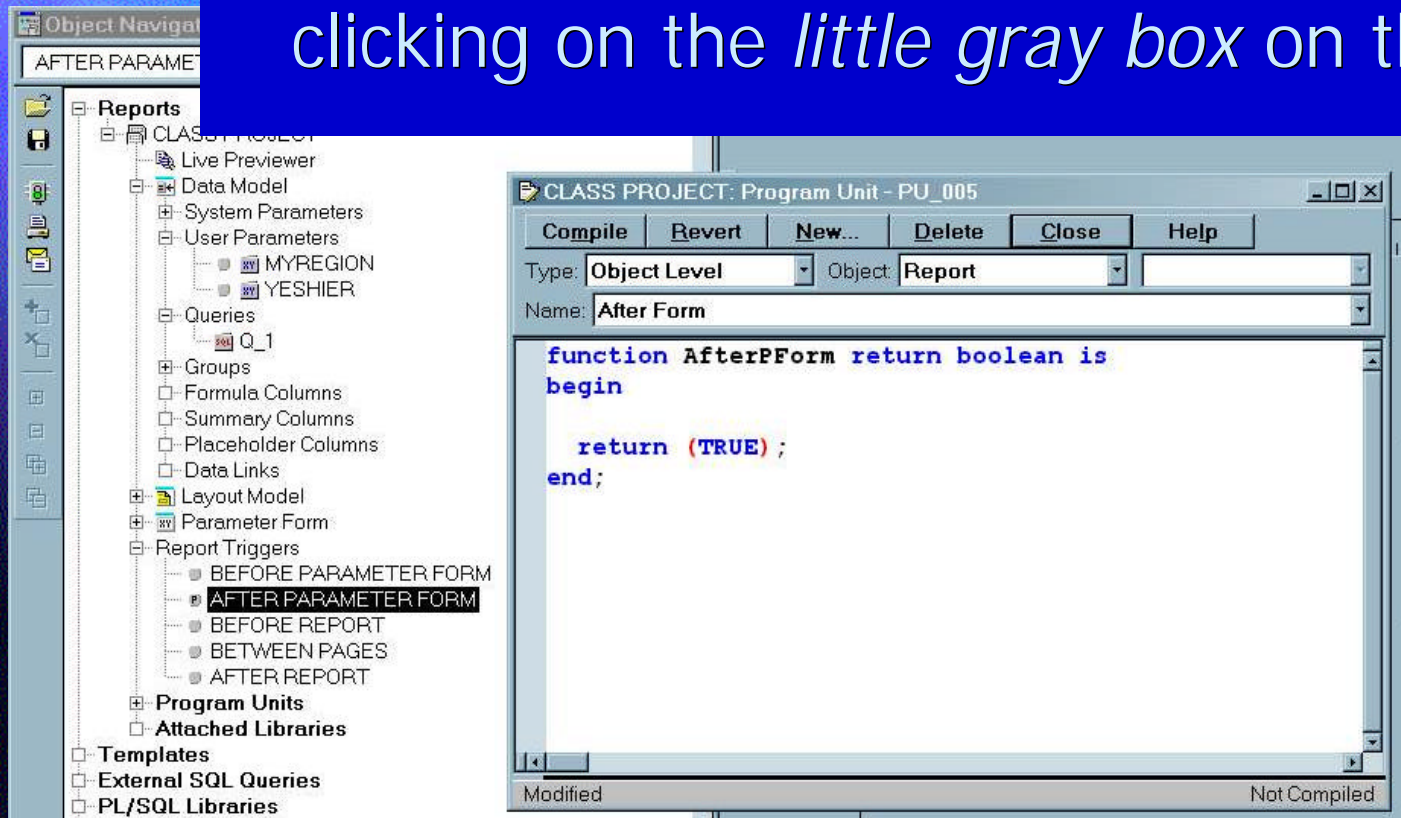
The screenshot shows the 'Parameter Form Builder' dialog box from the 'Report Builder for Windows 95 / NT' application. The dialog has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Format', 'Arrange', and 'Print'. It contains three text input fields: 'Title' with the text 'Counting the Universe', 'Hint Line' with the text 'Enter values for the parameters', and an empty 'Status Line' field. Below these is a table with two columns, 'Parameter:' and 'Label:'. The table lists several parameters: 'ORIENTATION' (Orientation), 'BACKGROUND' (Run in Background), 'MODE' (Output Mode), 'PRINTJOB' (Show Print Job Dialog), 'MYREGION' (2-digit Region:), and 'yesHier' (1=Hier, 0= nonHier). The 'MYREGION' and 'yesHier' rows are highlighted with a black background. At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.

Parameter:	Label:
ORIENTATION	Orientation
BACKGROUND	Run in Background
MODE	Output Mode
PRINTJOB	Show Print Job Dialog
MYREGION	2-digit Region:
yesHier	1=Hier, 0= nonHier

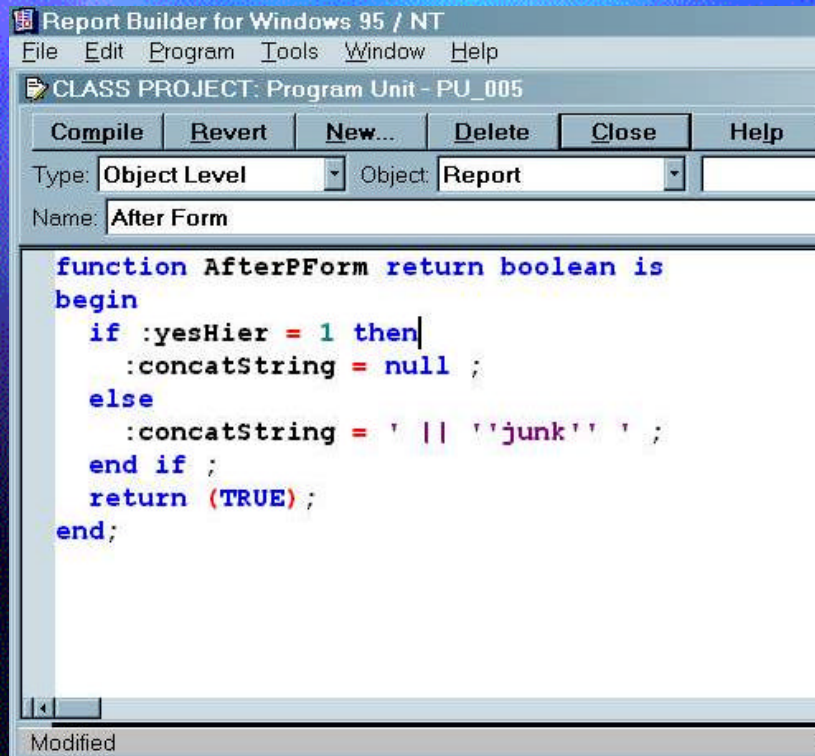
- Click the *traffic light* and notice that a place has been made for it on the Parameter Form, though it's a bit obscure.
- Before we fix that, click the *RED X* to NOT run the report, then *Okay*.
- Now run the *Parameter Form Builder* (under *Tools*). You'll get this screen if you scroll down.

The AfterParameterForm Trigger

- After using this *form builder* to change the labels, click *OK*.
- Now expand *Navigator, Report Triggers, AfterParameterForm*, and open it by double clicking on the *little gray box* on the left.



AfterParameterForm Logic



- Our logic will be this:
 - **if** the users says to do a Hier count **then** let the SQL proceed unchanged **else** change the sub-sub-query that gets IDs that are SUBJINSP so that it gets no IDs at all.
- We can accomplish the changing of the SUBJINSP query by changing its WHERE criteria into an impossible thing, like
 - ...where universeType = 'SUBJINSPjunk'
 - Nothing will match this and we will get a non-exclusionary, and so non-hier, count.
- So here in the AfterParameterForm code, let's create a variable that we can concatenate with the SUBJINSP term.

AfterParameterForm Error

- Compile it
- We get an error (*concatString* is not defined), so let's define it just like we did *yesHier*
- *Data Model, User Parameters, plusBox*, rename it, set its *Type* property to 'Character'

The screenshot displays the SAP Report Builder environment. The **Object Navigator** on the left shows the project structure: **Reports** > **CLASS PROJECT** > **Data Model** > **User Parameters**. Under **User Parameters**, the parameter **CONCATSTRING** is highlighted. The **Property Palette** on the right shows the properties for the selected parameter. The **Parameter** section is expanded, showing the **Datatype** property set to **Character**. The **CLASS PROJECT: Prog** window in the center shows the **After Form** event handler code, which defines the **concatString** function based on the **yesHier** parameter value.

Object Navigator

CONCATSTRING Find: []

Reports

- CLASS PROJECT
 - Live Previewer
 - Data Model
 - System Parameters
 - User Parameters
 - CONCATSTRING
 - MYREGION
 - YESHIER
 - Queries
 - Q_1
 - Groups
 - Formula Columns
 - Summary Columns
 - Placeholder Columns
 - Data Links
 - Layout Model
 - Parameter Form
 - Report Triggers

Property Palette

User Parameter: concatString

* General Information	
- Parameter	
Datatype	Character
Width	Character
Initial Value	Date
Validation Trigger	Number
List of Values	

CLASS PROJECT: Prog

Compile Revert

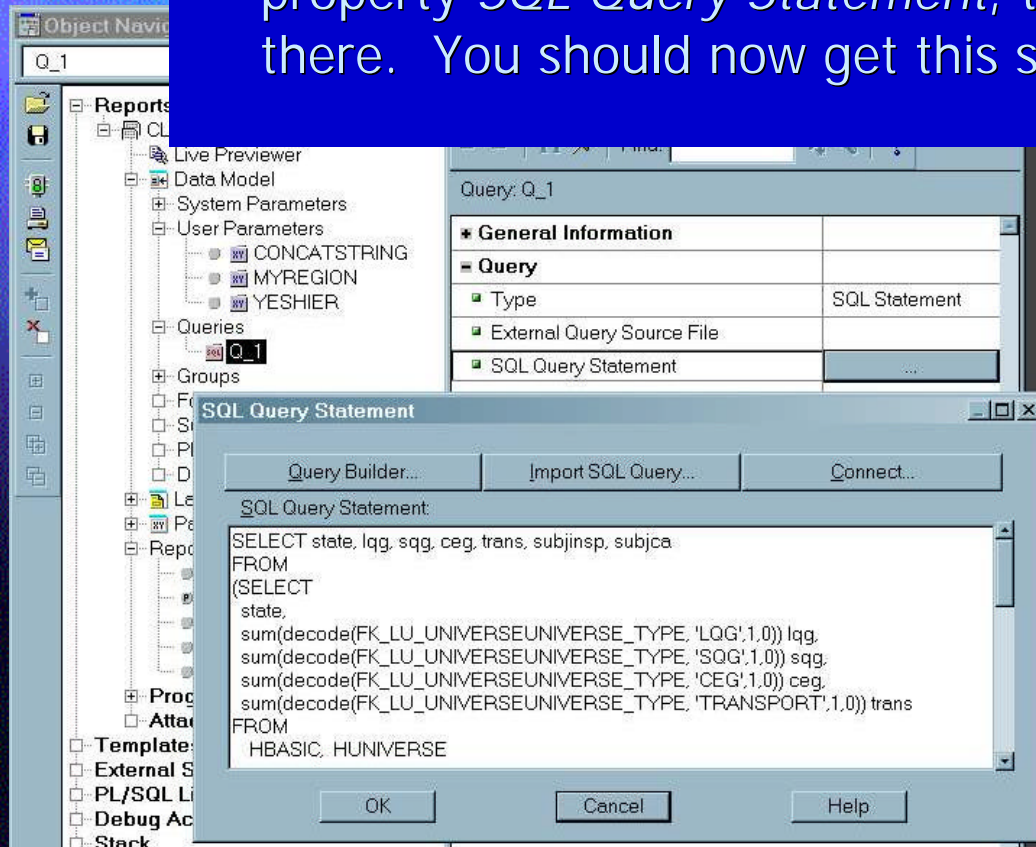
Type: Object Level

Name: After Form

```
function AfterPF
begin
  if :yesHier =
    :concatString
  else
    :concatString
  end if ;
  return (TRUE) ;
end
```


Yet Another Error

- Will it compile now? No, because the *equal sign* is for comparisons, not assignments. Instead we should use `:=`
- So alter both assignments, click *Compile* and the lack of an error means it's fine. Close that box.
- Now open the SQL, this time by clicking once on Q_1, then its property *SQL Query Statement*, then on the *button* that appears there. You should now get this screen.



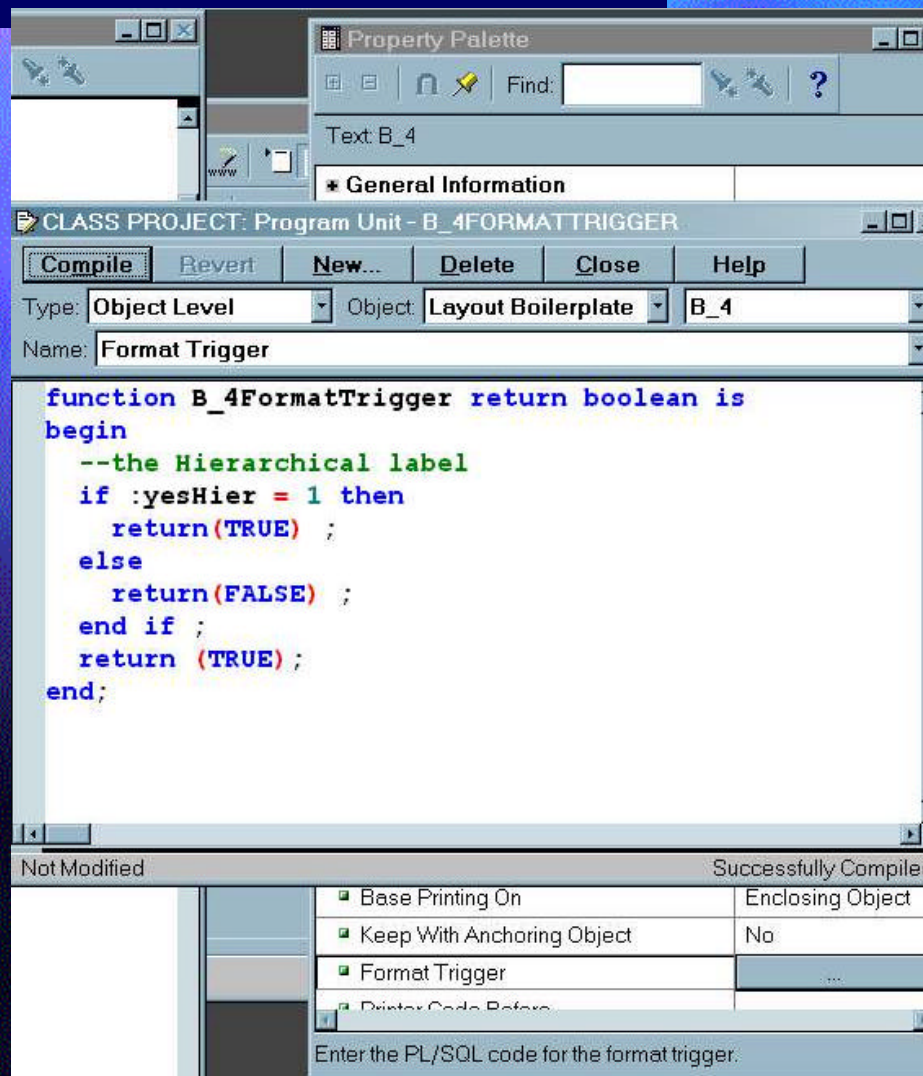
Final SQL Modifications

- Add the bit &concatString to the SQL.
- The ampersand simply inserts the value of concatString at that point in the code. If it's null, nothing changes, if its not null, no matches will be found.

SQL Que

```
(SELECT
state,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'LQG',1,0)) lqg,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SQG',1,0)) sqg,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'CEG',1,0)) ceg,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'TRANSPORT',1,0)) trans
FROM
HBASIC, HUNIVERSE
WHERE
REGION = :myRegion
AND FK_HBASICHANDLER_ID = HANDLER_ID
and handler_id not in ( select FK_HBASICHANDLER_ID from huniverse where
FK_LU_UNIVERSEUNIVERSE_TYPE = 'SUBJINSP'&concatString )
GROUP BY
state),
(SELECT
state secondState,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SUBJINSP',1,0)) subjinsp,
sum(decode(FK_LU_UNIVERSEUNIVERSE_TYPE, 'SUBJCA',1,0)) subjca
```

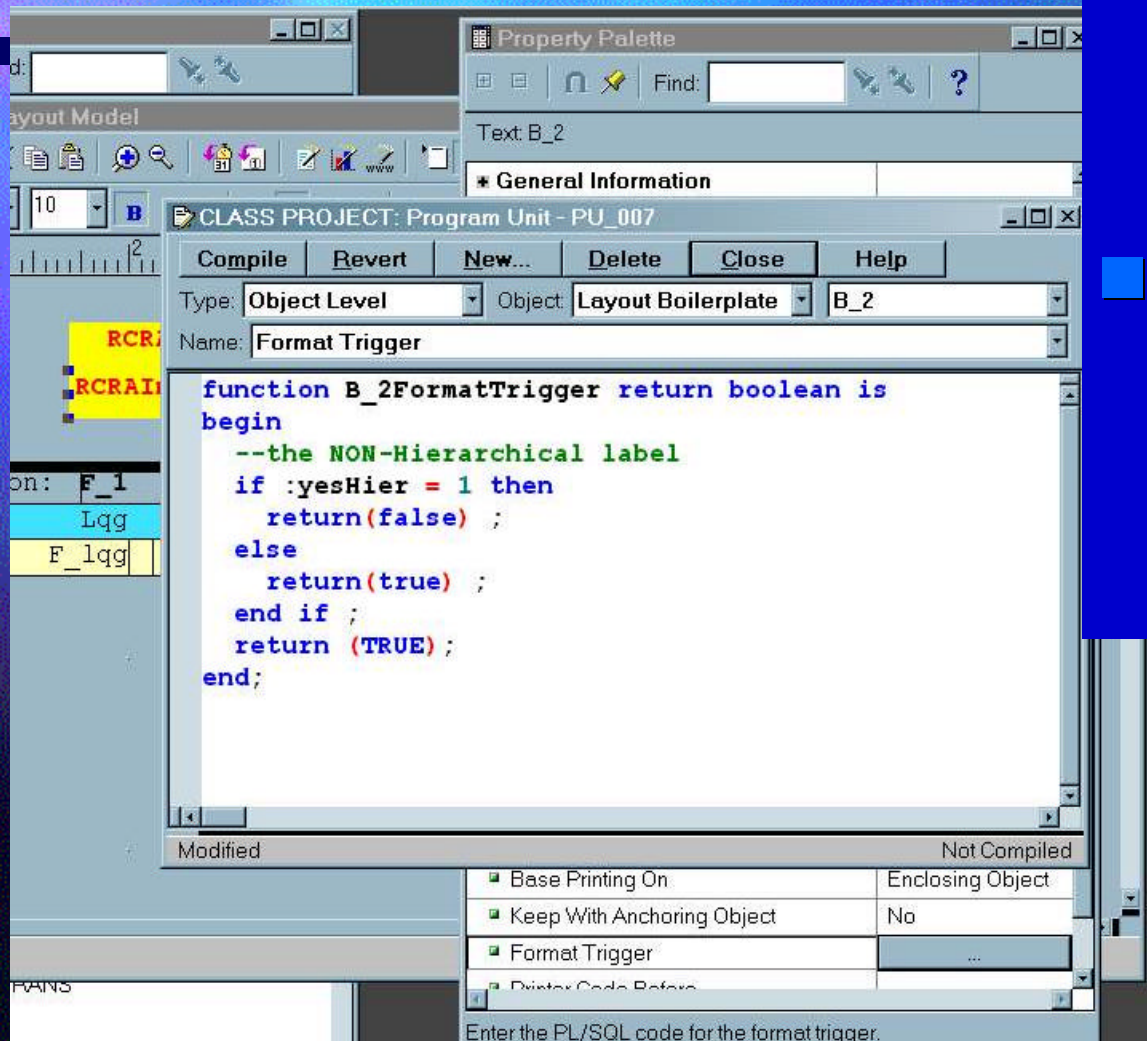

Label Format Triggers



- Run the code, try both type of counts and make sure it works. **It does!**
- Now lets make that label appear or disappear as appropriate.
- Our method is this:
 - Each label has a 'format trigger'. If the code in that trigger returns TRUE, it prints; if FALSE, it will not print.
- Get at the format triggers via the labels property palattes.

The Other Label Format Trigger

- Click Compile (remember your colons and semicolons!)
- For the Non-hier label, we do the opposite. You end up with this screen.



Finishing Touches

- Compile that, and close it.
- Now position one label over the other and hit the *green light*.
- By the way, you have been SAVING YOUR WORK, haven't you?

Report Builder for Windows 95 / NT

File Edit View Insert Format Arrange Program Tools Window Help

Object Navigator

B_4 Find:

Property Palette

class project: Report Editor - Live Previewer

Page: 1

Courier New Save 10 B I U

RCRAInfo Universe Count, Non-hierarchical

Report run on: 13-SEP-01

Region: 04

State	Lgg	Sgg	Ceg	Trans	Subjin
AL	287	1590	2846	99	
FL	499	16487	6082	339	
GA	454	3670	2937	197	
KY	382	532	1721	137	
MS	209	436	1989	116	
NC	515	2123	3881	84	
SC	411	1764	2176	185	
TN	528	870	1766	98	

Saves the active document

Margin

- T B_2
- T B_3
- T B_4

Enter the PL/SQL code for the format trigger.

It Works! The End!

Report Builder for Windows 95 / NT

File Edit View Insert Format Arrange Program Tools Window Help

Object Navigator

B_4 Find:

Property Palette

Find:

class project: Report Editor - Live Previewer

Page: 1

T Courier New Save 10 B I U

RCRAInfo Universe Count, Non-hierarchical

Report run on: 13-SEP-01

Region: 04

State	Lqg	Sqg	Ceg	Trans	Subjinsp	Subjca
AL	287	1590	2846	99	76	157
FL	499	16487	6082	339	82	136
GA	454	3670	2937	197	77	118
KY	382	532	1721	137	65	141
MS	209	436	1989	116	33	52
NC	515	2123	3881	84	90	108
SC	411	1764	2176	185	55	86
TN	528	870	1766	98	54	95

Saves the active document

Margin

- T B_2
- T B_3
- T B_4

Enter the PL/SQL code for the format trigger.

Appendix:

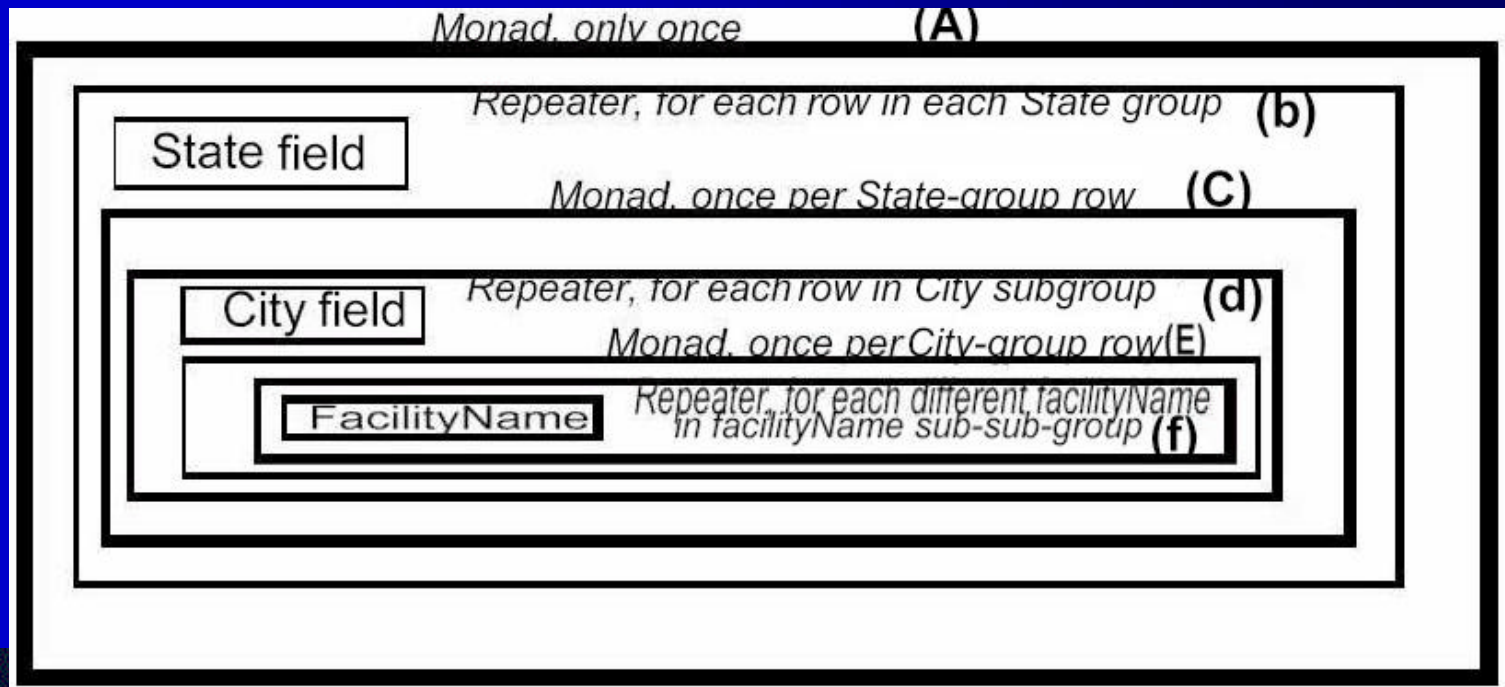
Dealing with Nested Frames

- Suppose our query gets back a table with columns named *State*, *City*, and *FacilityName*. Also, suppose our query is grouped by *State* and *City*.
- Since our query output is grouped by *State*, the output column *State* will have many entries but only a few unique values. Similarly, within each *State*, there will be many *facilityNames* grouped under duplicate *City* names.

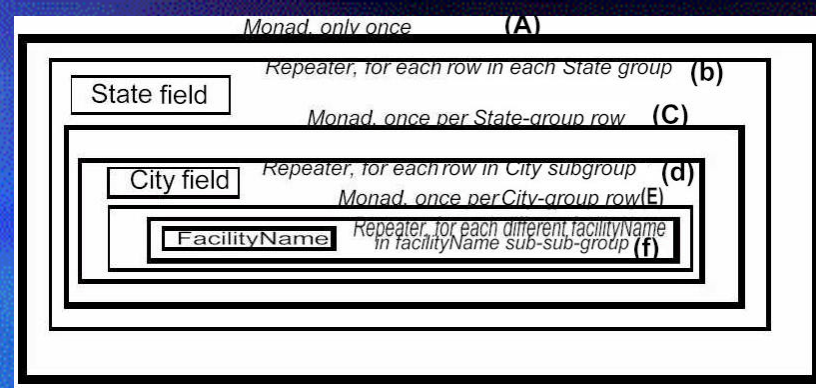
Appendix:

Dealing with Nested Frames

- A default Report generated by Report Builder will have the following nesting of frames:

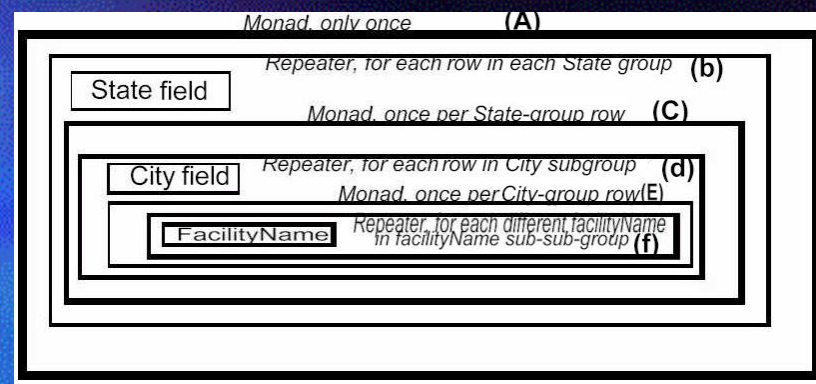


Dealing with Nested Frames



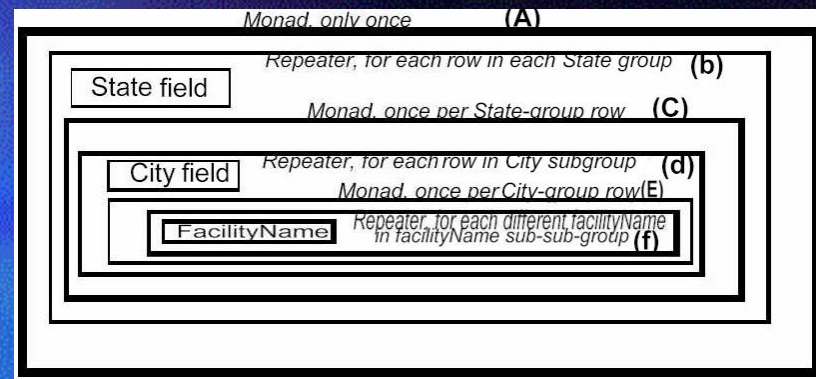
- The outermost frame, a Monad frame (A), is given the entire output table to print, but still the frame will only print once. This frame is actually unnecessary, but the Report Builder always wraps its reports inside one Monad frame.

Dealing with Nested Frames



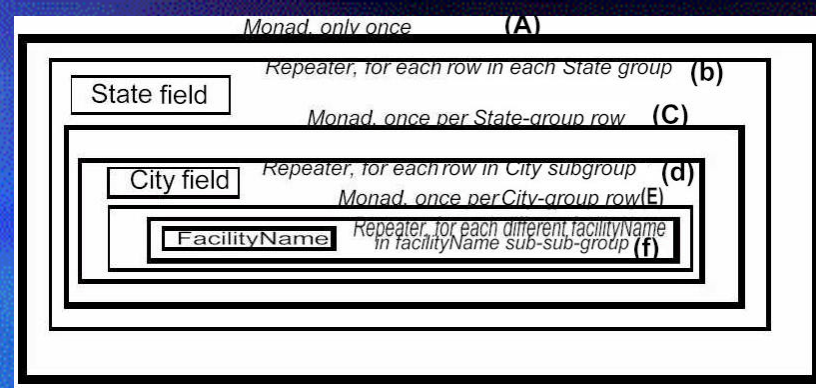
- Now consider the Repeater (b) inside of that. The frame (b) is "sourced" (i.e., "fed") by the grouping *G_State*. The group *G_State* is made up of the output table chopped up into chunks, each of which have identical values for *State*. (For example, the first chunk would be all the rows with *State* = 'AL'; the next chunk has all rows where *State* = 'FL'.) The frame (b) is given one chunk of this data to print, then it prints it, then it's given the next chunk, then the next, until each of the *G_State* groups have been printed.
- So once frame (b) prints all of its frame-contents using the first row of the first *G_State* chunk, it checks to see if there are any more rows in that *G_* group chunk; if not, it will start on the second chunk.

Dealing with Nested Frames



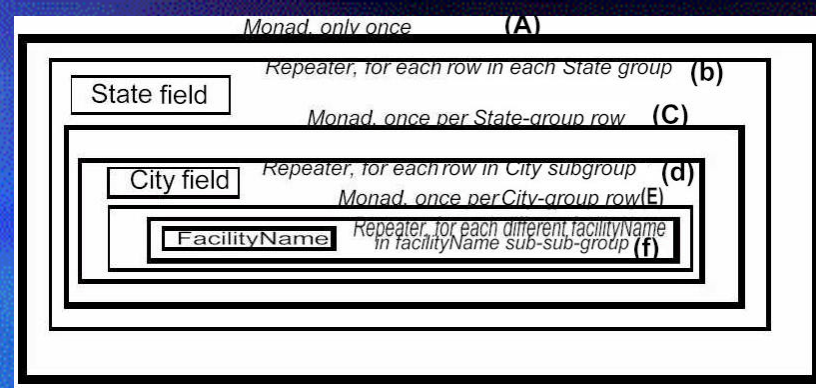
- But *before* it checks, it prints its contents, including Monad frame (C). Printing (C) simply causes the printing of the Repeater (d) to occur. Note that the only data (d) can “see,” however, is the first chunk of data that its parent frame is currently trying to print. In this case, that means the only data the *City* frame can see is the data for *one State*, and that data is itself now broken into chunks of identical values of *City*. The frame (d) now prints its contents using the first row of the first chunk, which means it prints the field *City* and the Monad frame (E).
- Printing (E) simply causes the printing of Repeating Frame (f), which is “sourced” by the group *G_facilityName*. This means that the frame (f) receives the first chunk of data (which is just one-State-and-one-City) and breaks it into chunks of identical *facilityNames*.

Dealing with Nested Frames



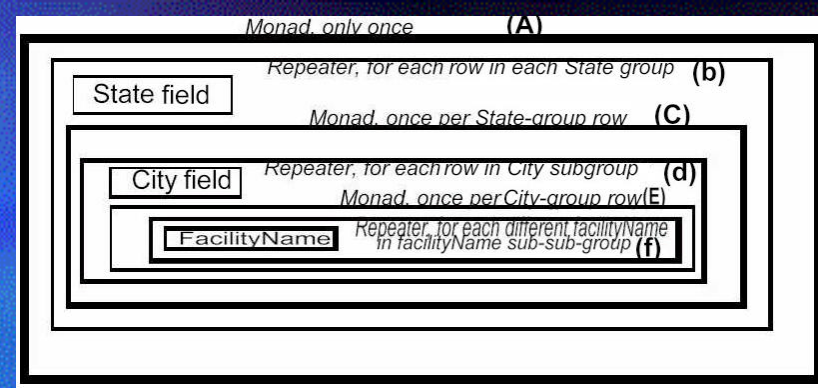
- Frame (f) then prints the first row of the first *facilityName* chunk, causing the printing of the field *facilityName*. It finishes with that row, and repeats it with the next, until the first chunk itself is done. (Remember, that chunk was just one *facilityName*, and probably just one row.) The frame then proceeds to the next *facilityName* chunk, and the next. With each printing, the output-table row-counter is advancing to the next row of data. Also, note that all this printing occurs only within the frame (f) and causes none of the other fields to be printed, *but the output table has only one row-counter, and it is being incremented each time (f) prints.*

Dealing with Nested Frames



- When the frame (f) is out of data, it has *also* finished moving through the contents of a one-State-and-one-City chunk. Therefore, when it has finished printing (and Repeating Frame (d) is finally given a chance to print again), the first chunk of same-city data has been completely gone through and used up, and the *next row available to frame (d) is the first row of the next one-State-and-one-City chunk*.
- So the next time the field *City* is printed, it has a new value, namely that of the second chunk of grouping data in *G_City*. In this way, each City's name is only printed *once*.

Dealing with Nested Frames



- Similarly, the next time that frame (b) is given a chance to print the value of *State* again, the row-counter in the output table will have passed the first State and now be in the second State's chunk of same-State data.

The End of the Appendix